

Document Version

Final published version

Licence

Dutch Copyright Act (Article 25fa)

Citation (APA)

Sergeyuk, A., Zakharov, I., Koshchenko, E., & Izadi, M. (2026). Human-AI experience in integrated development environments: a systematic literature review. *Empirical Software Engineering*, 31(3), Article 55.
<https://doi.org/10.1007/s10664-025-10793-0>

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

**Green Open Access added to [TU Delft Institutional Repository](#)
as part of the Taverne amendment.**

More information about this copyright law amendment
can be found at <https://www.openaccess.nl>.

Otherwise as indicated in the copyright section:
the publisher is the copyright holder of this work and the
author uses the Dutch legislation to make this work public.



Human-AI experience in integrated development environments: a systematic literature review

Agnia Sergeyuk¹ · Ilya Zakharov¹ · Ekaterina Koshchenko² · Maliheh Izadi³

Received: 10 March 2025 / Accepted: 15 December 2025

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2026

Abstract

The integration of Artificial Intelligence (AI) into Integrated Development Environments (IDEs) is reshaping software development, fundamentally altering how developers interact with their tools. This shift marks the emergence of Human-AI Experience in Integrated Development Environment (in-IDE HAX), a field that explores the evolving dynamics of Human-Computer Interaction in AI-assisted coding environments. Despite rapid adoption, research on in-IDE HAX remains fragmented, which highlights the need for a unified overview of current practices, challenges, and opportunities. To provide a structured overview of existing research, we conduct a systematic literature review of 90 studies, summarizing current findings and outlining areas for further investigation. We organize key insights from reviewed studies into three aspects: Impact, Design, and Quality of AI-based systems inside IDEs. Impact findings show that AI-assisted coding enhances developer productivity but also introduces challenges, such as verification overhead and over-reliance. Design studies show that effective interfaces surface context, provide explanations and transparency of suggestion, and support user control. Quality studies document risks in correctness, maintainability, and security. For future research, priorities include productivity studies, design of assistance, and audit of AI-generated code. The agenda calls for larger and longer evaluations, stronger audit and verification assets, broader coverage across the software life cycle, and adaptive assistance under user control.

Keywords Human-computer interaction · Artificial intelligence · Integrated development environment · Programming · User studies · User experience

1 Introduction

The integration of Artificial Intelligence (AI) into Integrated Development Environments (IDEs) has gained significant traction in recent years, driven by the promise of enhanced developer productivity, improved software quality, and more efficient workflows (Ziegler

Communicated by: Yu Huang and Antonino Ferraro.

Extended author information available on the last page of the article

et al. 2022; Izadi et al. 2024). Recent large-scale industry surveys show that 76% of respondents are either using or planning to use AI tools in their development process (Stack Overflow 2024). Additionally, more than 50% of respondents note such benefits as reduced time spent searching for information, faster coding and development, quicker completion of repetitive tasks, and an overall increase in productivity (JetBrains 2024).

Traditionally, Human-Computer Interaction (HCI) has focused on how users interact with software and tools. With AI now integrated into development workflows, this focus is shifting toward Human-AI Experience (HAX). Unlike conventional software, AI acts as an active collaborator rather than just a tool, changing how developers work and interact with their environment.

Given the growing reliance on AI-enhanced IDEs in software engineering (e.g., Cursor¹), it is crucial to merge and evaluate existing evidence in the field of in-IDE HAX. Doing so will help to identify recurring themes and gaps in the literature regarding the developer experience with AI assistance.

Recent literature reviews have provided comprehensive overviews of how Large Language Model (LLM) support software engineering tasks: summarizing model capabilities, task coverage, and evaluation techniques (He et al. 2025; Zhou et al. 2025a; Hou et al. 2024; Durrani et al. 2024). However, these reviews do not analyze the interaction experience inside developer workspaces, where the software development lifecycle takes place. Our work is, to the best of our knowledge, the first systematic review of in-IDE HAX. This lens highlights experience-level patterns and design challenges that general and capability-focused reviews on LLM in software engineering do not capture.

This work builds on our earlier survey of in-IDE HAX (Sergeyuk et al. 2024), which provided an initial exploration of the topic and highlighted its potential. In contrast, the present study is extended in both scope and methodological rigor. It applies a formal Systematic Literature Review (SLR) process, following the Preferred Reporting Items for Systematic reviews and Meta-Analyses (PRISMA) framework (Moher et al. 2010). Compared to the 36 papers in the initial study, which did not follow a systematic protocol, it includes 90 papers identified through a multi-step procedure that combines database search, backward snowballing, and expert-informed additions. Beyond expanding the dataset, this review introduces several analytical contributions. It not only organizes findings along three key research dimensions: Design, Impact, and Quality. It also provides a detailed analysis of the study context, empirical methods, and proposed future directions. These contributions position this study not as a follow-up review, but as a foundational reference for cumulative and reproducible progress in the research field of in-IDE HAX.

Using our protocol, we identified 257 studies and thoroughly reviewed 90 that were relevant to in-IDE HAX. The goals of this review are to (a) identify common themes and patterns in the literature, (b) merge the current state of knowledge regarding in-IDE HAX, and (c) identify critical gaps that can guide future research efforts. We focus on how the presence of an AI assistance changes what happens in the very place where code is written, inspected, or tested, rather than how AI helps in software engineering broadly.

Our review reveals the focus of in-IDE HAX research activity on professional software development settings, examining the code implementation stage of the Software Development Lifecycle (SDLC), often using GitHub Copilot as the primary example. Other lifecycle stages, such as requirements, testing, and deployment, receive limited attention.

¹Cursor: The AI Code Editor <https://www.cursor.com/>

Educational contexts and perspectives of non-users or people who have stopped using AI assistance are also rarely studied.

According to our review, key findings in the field span three core dimensions: *Impact* (effects on developers, tasks, and workflows), *Design* (how AI is integrated into the IDE and interacted with), and *Quality* (properties of AI outputs such as correctness, security, and readability). *Impact*-oriented studies (74/90) report productivity improvements, especially among experienced users, but also note increased time spent on result verification and concerns about over-reliance. Studies in the *Design* dimension (28 out of 90 papers) examine autocompletion and conversational systems, with some exploring emerging hybrid approaches. Research highlights the influence of prompt structure and interface properties such as context awareness, transparency, and user control on experience with AI tooling. In the *Quality* dimension (19/90), studies evaluate correctness, maintainability, and security of generated code, often calling for improved validation and diagnostic support.

Future research directions identified in the literature focus on several key areas, including productivity factors, audit mechanisms for AI-generated code, and the design of AI assistance tools. There is a strong emphasis on personalization and transparency, with a growing interest in exploring the long-term adoption of AI tools. Methodologically, studies suggest the need for larger-scale evaluations, comparative research across user groups and contexts, and longitudinal studies to track the evolution of AI's impact over time. Despite significant progress, many research directions remain underexplored, particularly in areas such as AI governance, user control, and proactivity.

In summary, while substantial progress has been made in understanding in-IDE HAX, key gaps remain, particularly in exploring non-user perspectives and underrepresented stages of the software development lifecycle. Future research should focus on broadening the contexts studied, incorporating longitudinal designs, and refining methodologies to provide a more comprehensive understanding of AI's long-term impact and its role across diverse user groups.

Our study offers the following contributions:

- **Comprehensive synthesis of existing research.** We analyze 90 studies, categorizing them based on their research goals, methodological approaches, and key findings. This synthesis reveals dominant research themes, study contexts, and trends in HAX research.
- **Characterization of HAX research dimensions.** We classify research efforts into three core aspects: (a) **Impact**, examining how AI affects developers' workflows, productivity, and experience; (b) **Design**, focusing on how AI assistance is integrated into IDEs; and (c) **Quality**, assessing AI-generated code for security, comprehensibility, and adequacy.
- **Identification of future research opportunities.** By analyzing the proposed future work in the studies, we identify under-explored topics, such as AI governance, user control, and proactivity.

By addressing these aspects, we provide a structured overview of in-IDE HAX research, informing both academic and industry efforts to design and evaluate AI-powered development tools.

2 Method

To answer our research questions, we set out to conduct a systematic literature review in accordance with the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) framework (Moher et al. 2010).

The following Research Questions (RQs) guided our literature review:

- **RQ 1. What aspects of software development related to HAX within IDEs have been extensively studied, and which areas remain under-explored?** To address RQ1, we defined the stages of SDLC according to the Waterfall model (Alshamrani and Bahattab 2015). The Waterfall model was chosen for its clear delineation of development stages, which facilitated systematic categorization. We also examined study contexts and three key aspects of HAX identified by a previous literature review (Sergeyuk et al. 2024).
- **RQ 2. What are the key findings in the field of HAX in IDEs?** To answer RQ2, we systematically examined goals, research questions, and key findings from published reports. That allowed us to determine key research themes in the field and define methodological trends.
- **RQ 3. What are the potential directions for future research and development in the field of HAX in IDEs?** To address RQ3, we analyzed the lists of future research directions proposed in the reviewed studies, identifying recurring themes and uncovering how studies build on each other's findings.

To address RQ1 and RQ2, we applied an in-IDE HAX taxonomy (Sergeyuk et al. 2024) developed through a staged qualitative procedure. Two authors independently conducted open coding of the corpus to propose candidate categories, followed by consensus meetings to merge or split codes, assign names, and draft a codebook with definitions and examples. The codebook was piloted on a subset, refined, and then used to code the full dataset by two authors, with disagreements adjudicated by a third. Categories with low support or conceptual overlap were consolidated to improve parsimony while preserving coverage. The resulting structure comprises three distinct dimensions: *Impact* (effects on developers, tasks, and workflows), *Design* (how AI is integrated into and interacted with in the IDE), and *Quality* (properties of AI outputs such as correctness, security, and readability).

2.1 Planning

Based on the best practices established in the field (Moher et al. 2010; Zhou et al. 2015; Carrera-Rivera et al. 2022; Kitchenham et al. 2010, 2009), we defined eligibility criteria to ensure a comprehensive and methodologically sound review. These criteria were guided by the research objectives and scope of this review. We report our protocol details below:

- **Timeframe:** Studies published between 2022 and 2024 were included to ensure relevance to the advancements in AI-driven IDE features.
- **Language:** Only studies published in English were included.
- **Publication Status:** The review included peer-reviewed journal articles and conference proceedings. Recognizing the need to account for the rapid development in this field,

preprints and unpublished dissertations were also considered if they provided unique insights directly relevant to the research objectives.

- **Population:** We considered studies in the field of computer science that focus on software developers or users engaging with AI-enhanced tools within IDEs.
- **Intervention:** Studies were included if they investigated in-IDE HAX.
- **Comparison:** Not applicable.
- **Outcomes:** This review included studies that presented empirical findings, whether qualitative, quantitative, or mixed-methods.
- **Study Design:** The review included original empirical studies.

IDE in this review denotes any workspace that simultaneously offers (a) code editing, (b) immediate execution or preview, and (c) contextual AI feedback in the same window. Traditional desktop IDEs (IntelliJ, VS Code) and notebook-style environments (Jupyter, Databricks) satisfy all three criteria; plug-ins that embed a full editor and run code inline also qualify. Tools that push AI suggestions outside the primary editing surface (*e.g.*, web explainers, offline static analysers) are excluded.

During the planning stage, as recommended by the guidelines (Centre for Reviews and Dissemination (UK) 1995), we established quality assessment criteria for studies that would be included after the initial screening phase. Our quality assessment questions were inspired by the recommendations of Zhou et al. (2015):

- **Reporting:** Is there a clear statement of the aims of the research?
- **Rigor:** Is the study design clearly defined?
- **Credibility:** Is there a clear statement of findings related to the aims of research?
- **Relevance:** Is the study of value for research or practice?

Each question was accompanied by the following scale: 0 — No, and not considered; 0.5 — Partially; 1 — Yes. Following recommended guidelines, we set the cutoff for further inclusion in the review at a score higher than 2. These criteria ensured a systematic evaluation of the research's aims, study design, methodological soundness, value and validity of findings.

Furthermore, the data extraction form was defined during the planning stage. This form was guided by our research questions and the goal of assessing the representativeness of the included studies, given their varying venues and scopes. Therefore, in addition to the name of each report, the extraction form included the following fields; authors names and affiliation, DOI, publication date, venue, goal of the study, research questions, key findings, future work, scope (based on the taxonomy defined in previous work Sergeyyuk et al. 2024), and SDLC stage.

To ensure coverage of a diverse range of scholarly sources, we selected several well-known digital libraries for our initial search. These included *ACM Digital Library*, *DBLP*, *IEEE Digital Library*, *ISI Web of Science*, *ScienceDirect*, *Scopus*, *Springer Link*, and *arXiv*. We included *arXiv* to capture insights from emerging fields, recognizing that valuable information can often be found in non-peer-reviewed papers. Additionally, given the prevalence of positive results in published literature, we included preprints and unpublished dissertations to reduce the risk of excluding studies with negative or null findings.

2.2 Identifying and Screening

In November 2024, after finishing the planning phase, we searched the aforementioned sources, using titles, keywords, and abstracts as the basis for our queries. We optimized for high recall at the search stage and enforced precision during manual screening.

The search string has a two-part structure: a core query covering terminology related to AI assistants and Human-AI Experience (S_{core}), optionally combined with a query specifying IDE-related terms (S_{ide}).

- S_{core} : *“AI Assistant” OR “AI Companion” OR “AI-Powered Programming Tool” OR “Code Completion Tool” OR “Coding Assistant” OR “Copilot” OR “Intelligent Code Assistant” OR “LLM-Based Coding Assistant” OR “LLM-Powered Coding Assistant” OR “LLM4Code” OR “Programming Assistant” OR “Human-AI Experience” OR “Human-AI Co-Creation” OR “Human-AI Collaboration” OR “Human-AI Interaction”*
- S_{ide} : *“Integrated Development Environment” OR “Code Editor” OR “Coding Environment” OR “Development Environment” OR “Programming Environment”*

The full query was used in IEEE Xplore, Web of Science, ScienceDirect, Scopus, Springer Link, and arXiv. In ACM Digital Library and DBLP, we used only S_{core} due to limited filtering capabilities and higher baseline relevance. We did not include “developer” as a required keyword, since it tended to retrieve general developer studies and exclude relevant work focused on tools or interfaces that met our criteria but were not indexed using that term. All search string variations and their application per library are available in the Supplementary materials (Sergeyuk et al. 2025).

We complemented our database search with two additional inclusion strategies. First, we included 35 studies from our prior literature survey (Sergeyuk et al. 2024), which, although not systematic, provided coverage of early work in this fast-moving field. Second, we manually added 30 relevant studies known to the authors through domain expertise. These were identified as frequently cited or influential in recent research but were missed due to inconsistent metadata, terminology drift, or indexing limitations in digital libraries. This expert-based supplementation is recommended in SLR methodology (Wohlin 2014; Kitchenham et al. 2009) to reduce omission of contextually critical work. All manually included studies were subjected to the same quality assessment and eligibility criteria as search-derived papers to ensure methodological consistency.

All retrieved records were manually screened against our in-IDE HAX inclusion criterion. This step preserved recall while restoring precision. In total, 223 papers entered the screening phase of the systematic literature review. We identified and excluded 30 duplicates appearing across multiple databases. Next, we conducted an initial screening, applying the exclusion criteria and identifying 114 papers out of the scope of the current SLR.

2.3 Assessing Eligibility

In the next stage of the systematic literature review, we assessed the eligibility of 79 full-text articles based on their quality. As described in Subsection 2.1, we applied four evaluation

criteria: Reporting, Rigor, Credibility, and Relevance. Articles were accepted only if their combined score on these criteria exceeded 2.

The quality assessment was conducted independently by the second and third authors, followed by a discussion to reconcile any discrepancies. In cases of disagreement, the first author acted as an arbitrator. This structured process ensured consensus among the reviewers for the final quality assessment. A total of 8 papers were excluded due to insufficient quality scores lower or equal to 2, dictated by insufficient relevance and rigor. Excluded papers were accessed by authors as only partially or not sufficient in terms of clarity of the statement of the aims of the research, its design, findings related to the aims of the research, and value for research or practice.

2.4 Backward Snowballing

To complement the systematic search, we performed backward snowballing (Wohlin 2014) on papers deemed eligible for inclusion. First, we extracted citations from these papers and screened them against the report-related criteria outlined in Subsection 2.1. References that did not meet these criteria were excluded. We next removed any papers already identified in previous steps. We assessed the remaining studies against the study-level inclusion criteria described in Subsection 2.1, excluding out-of-scope papers. After the first round of this process, 18 new in-scope papers were identified. A second round of snowballing did not yield any additional relevant papers. These 18 studies underwent the eligibility assessment process outlined in Subsection 2.3.

During the revision phase, we re-verified the included studies. This process led to the removal of 2 papers (one out-of-scope, one duplicate) and the addition of 3 previously overlooked studies, ensuring that the final set reflects both the inclusion criteria and recent relevant literature and yielding a final total of 90 papers to be included in the review.

Note that eligibility was determined by the date of first public availability. We included studies first available between January 2022 and November 2024, including preprints on arXiv. When a study later appeared in an archival venue in 2025, we cite the archival version with its DOI and venue year. As a result, some entries list 2025 in the reference list while remaining within the time window of our review based on the preprint date.

2.5 Data Extraction

As mentioned in Subsection 2.1, from all 90 eligible papers, we extracted data about *Authors*, *Authors' Affiliation*, *DOI*, *Publication Date*, *Venue*, *Goal of the Study*, *Research Questions*, *Key Findings*, *Future Work with its topic and methodology*, *Scope*, *SDLC Stage*. In the later stages of data analysis, we added a *Context* column to indicate whether HAX was studied in a professional or educational setting. Moreover, we classified studies based on the types of tasks investigated in the work.

This work was conducted manually by the first author, with support from an AI-based system called Elicit². This tool is designed to facilitate the extraction and summarization of scientific papers. It supported this work by formulating concise descriptions of each study's goal, key findings, and future work. To ensure data integrity and accuracy, all tool-generated outputs were rigorously reviewed and cross-verified against the original papers. Information

²Elicit: The AI Research Assistant <https://elicit.com/>

was included into the final review only after confirming its faithful representation of the original meanings. This hybrid approach ensured both efficiency and precision in the data extraction process.

Data extraction yielded a comprehensive table with 90 studies in it. We provide the full table in Supplementary materials (Sergeyuk et al. 2025), and present a shortened overview of the studies in Table 1.

3 Results

This section presents key findings from our review of the studies on in-IDE HAX. We analyze the contexts of the studies, the reported impact of AI on developers, the design of AI tools for coding, and the quality of the AI tools. The expanded corpus confirms the adequacy of the initial three-part framing (Sergeyuk et al. 2024) and adds finer subscopes within each dimension, but it does not introduce any new top-level dimensions. We also examine future work directions proposed in the reviewed literature and methodological patterns of the studies. As described in Fig. 1 and Section 2, we screened a total of 256 papers, 100 of which were assessed for eligibility, with 90 included in the final review stage.

3.1 Context of Studies

The studies analyzed in this review represent two primary contexts for working with AI in software development: *professional* (71 out of 90) and *educational* (19/90) (see Fig. 2 for paper IDs). These contexts provide contrasting perspectives on the in-IDE integration of AI tools, with professional studies focusing on real-world applications and educational studies exploring their role in learning environments. Note that studies in educational settings may be underrepresented because our scope targets in-IDE contexts and students do not always work inside full IDEs during coursework. For a literature review of broader AI tools for coding in a strictly educational context, we refer the reader to (Agbo et al. 2025; Ariza et al. 2025)

The majority of professional studies emphasize the practical utility of AI tools. Studies in this context address coding (17/71) and maintenance (8/71), with limited exploration of requirements gathering (1/71). However, 46 out of 71 studies do not specify a target SDLC stage, instead evaluating AI tooling for general programming use without stage-level framing, indicating a gap in contextual precision.

Educational studies primarily focus on pedagogical insights. These studies often investigate how students use AI tools to learn programming or solve computational problems. While a few studies address coding (6/16) and requirements formulation (1/16), most (13/16) do not specify the SDLC stage, aligning with the broader exploratory nature of educational research.

The distribution depicted on Fig. 3, reveals notable gaps in the literature:

- Lack of perspectives from people who do not use AI tools for development for a variety of reasons.

Table 1 List of studies included in the review. *Dating note.* Inclusion used first public availability between January 2022 and November 2024. We cite the archival version with its DOI and venue year. Some entries show 2025 because an earlier preprint falls within this window

ID	Title	Authors
1	An Empirical Evaluation of GitHub Copilot's Code Suggestions	Nguyen and Nadi (2022)
2	Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions.	Pearce et al. (2022)
3	Assessing the quality of GitHub copilot's code generation	Yetistiren et al. (2022)
4	Better Together? An Evaluation of AI-Supported Code Translation	Weisz et al. (2022)
5	Designing PairBuddy—A Conversational Agent for Pair Programming	Robe and Kuttal (2022)
6	Discovering the Syntax and Strategies of Natural Language Programming with Generative Language Models	Jiang et al. (2022)
7	Documentation Matters: Human-Centered AI System to Assist Data Science Code Documentation in Computational Notebooks	Wang et al. (2022)
8	Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models	Vaithilingam et al. (2022)
9	Exploring the Learnability of Program Synthesizers by Novice Programmers	Jayagopal et al. (2022)
10	Github copilot in the classroom: learning to code with AI assistance	Puryear and Sprint (2022)
11	How Readable is Model-generated Code? Examining Readability and Visual Inspection of GitHub Copilot	Al Madi (2022)
12	Investigating Explainability of Generative AI for Code through Scenario-based Design	Sun et al. (2022)
13	Is GitHub Copilot a Substitute for Human Pair-Programming? An Empirical Study	Imai (2022)
14	Practitioners' expectations on automated code comment generation	Hu et al. (2022)
15	Productivity Assessment of Neural Code Completion	Ziegler et al. (2022)
16	Taking Flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools	Bird et al. (2022)
17	"It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers	Prather et al. (2023)
18	A Case Study in Engineering a Conversational Programming Assistant's Persona	Ross et al. (2023b)
19	A Case Study on Scaffolding Exploratory Data Analysis for AI Pair Programmers	Zhou and Li (2023)
20	A Mixed Reality Approach for Innovative Pair Programming Education with a Conversational AI Virtual Avatar	Manfredi et al. (2023)
21	AI-Powered Chatbots and the Transformation of Work: Findings from a Case Study in Software Development and Software Engineering	Süße et al. (2023)
22	Anticipating User Needs: Insights from Design Fiction on Conversational Agents for Computational Thinking	Penney et al. (2023)
23	Case Study: Using AI-Assisted Code Generation In Mobile Teams	Vasiliniuc and Groza (2023)
24	CoLadder: Supporting Programmers with Hierarchical Code Generation in Multi-Level Abstraction	Yen et al. (2023)
25	Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language	Denny et al. (2023)
26	Copilot for Xcode: Exploring AI-Assisted Programming by Prompting Cloud-based Large Language Models	Tan et al. (2023)

Table 1 (continued)

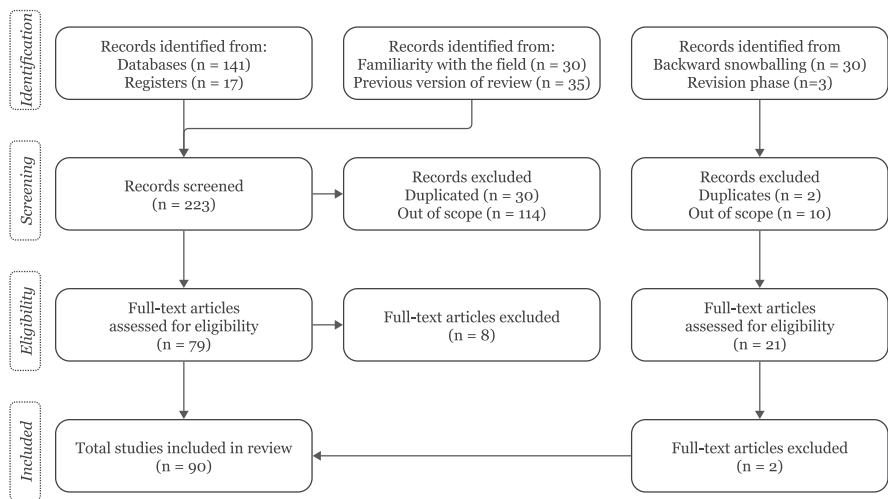
ID	Title	Authors
27	From “Ban It Till We Understand It” to “Resistance is Futile”: How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot	Lau and Guo (2023)
28	Grounded Copilot: How Programmers Interact with Code-Generating Models	Barke et al. (2023)
29	How Novices Use LLM-Based Code Generators to Solve CS1 Coding Tasks in a Self-Paced Learning Environment	Kazemitabaar et al. (2023b)
30	In-IDE Generation-based Information Support with a Large Language Model	Nam et al. (2023)
31	Is GitHub’s Copilot as Bad as Humans at Introducing Vulnerabilities in Code?	Asare et al. (2023)
32	Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants	Sandoval et al. (2023)
33	On the Design of AI-powered Code Assistants for Notebooks	McNutt et al. (2023)
34	On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot	Mastropaolo et al. (2023)
35	Practices and Challenges of Using GitHub Copilot: An Empirical Study	Zhang et al. (2023)
36	Practitioners’ Expectations on Code Completion	Wang et al. (2023a)
37	Slide4N: Creating Presentation Slides from Computational Notebooks with Human-AI Collaboration	Wang et al. (2023b)
38	Spellburst: A Node-based Interface for Exploratory Creative Coding with Natural Language Prompts	Angert et al. (2023)
39	Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming	Kazemitabaar et al. (2023a)
40	The Impact of AI on Developer Productivity: Evidence from GitHub Copilot	Peng et al. (2023)
41	The Programmer’s Assistant: Conversational Interaction with a Large Language Model for Software Development	Ross et al. (2023a)
42	Towards More Effective AI-Assisted Programming: A Systematic Design Exploration to Improve Visual Studio IntelliCode’s User Experience	Vaithilingam et al. (2023)
43	Using GitHub Copilot to Solve Simple Programming Problems	Wermelinger (2023)
44	“It would work for me too”: How Online Communities Shape Software Developers’ Trust in AI-Powered Code Generation Tools	Cheng et al. (2024a)
45	A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges	Liang et al. (2024)
46	A Study on Developer Behaviors for Validating and Repairing LLM-Generated Code Using Eye Tracking and IDE Actions	Tang et al. (2024)
47	A Transformer-Based Approach for Smart Invocation of Automatic Code Completion	de Moor et al. (2024)
48	A User-centered Security Evaluation of Copilot	Asare et al. (2024)
49	AI Tool Use and Adoption in Software Development by Individuals and Organizations: A Grounded Theory Study	Li et al. (2024)
50	An Analysis of the Costs and Benefits of Autocomplete in IDEs	Jiang and Coblenz (2024)
51	An Empirical Study of Code Search in Intelligent Coding Assistant: Perceptions, Expectations, and Directions	Liu et al. (2024)
52	An Empirical Study on Usage and Perceptions of LLMs in a Software Engineering Project	Rasnayaka et al. (2024)
53	An Exploratory Study on Upper-Level Computing Students’ Use of Large Language Models as Tools in a Semester-Long Project	Tanay et al. (2024)
54	Analyzing Prompt Influence on Automated Method Generation: An Empirical Study with Copilot	Fagadau et al. (2024)

Table 1 (continued)

ID	Title	Authors
55	Ansible Lightspeed: A Code Generation Service for IT Automation	Sahoo et al. (2024)
56	Are Prompt Engineering and TODO Comments Friends or Foes? An Evaluation on GitHub Copilot	O'Brien et al. (2024)
57	Can Developers Prompt? A Controlled Experiment for Code Documentation Generation	Kruse et al. (2024)
58	Copilot Evaluation Harness: Evaluating LLM-Guided Software Programming	Agarwal et al. (2024)
59	CoPrompt: Supporting Prompt Sharing and Referring in Collaborative Natural Language Programming	Feng et al. (2024)
60	Defendroid: Real-time Android code vulnerability detection via blockchain federated neural network with XAI	Senanayake et al. (2024)
61	Design Principles for Collaborative Generative AI Systems in Software Development	Chen and Zacharias (2024)
62	Developers' Perspective on Today's and Tomorrow's Programming Tool Assistance: A Survey	Kuang et al. (2024)
63	Evaluating Human-AI Partnership for LLM-based Code Migration	Omidvar Tehrani et al. (2024)
64	Exploring Interaction Patterns for Debugging: Enhancing Conversational Capabilities of AI-assistants	Chopra et al. (2024)
65	How Do Data Analysts Respond to AI Assistance? A Wizard-of-Oz Study	Gu et al. (2024)
66	IDA: Breaking Barriers in No-code UI Automation Through Large Language Models and Human-Centric Design	Shlomov et al. (2024)
67	Identifying the Factors That Influence Trust in AI Code Completion	Brown et al. (2024)
68	Improving Steering and Verification in AI-Assisted Data Analysis with Interactive Task Decomposition	Kazemitabaar et al. (2024a)
69	Investigating and Designing for Trust in AI-powered Code Generation Tools	Wang et al. (2024)
70	Investigating Interaction Modes and User Agency in Human-LLM Collaboration for Domain-Specific Data Analysis	Guo et al. (2024)
71	Ivie: Lightweight Anchored Explanations of Just-Generated Code	Yan et al. (2024)
72	Methodology for Code Synthesis Evaluation of LLMs Presented by a Case Study of ChatGPT and Copilot	Ságodi et al. (2024)
73	Multi-line AI-assisted Code Authoring	Dunay et al. (2024)
74	Non-Expert Programmers in the Generative AI Future	Feldman and Anderson (2024)
75	Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation	Gardella et al. (2024)
76	Prompt Sapper: A LLM-Empowered Production Tool for Building AI Chains	Cheng et al. (2024b)
77	Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming	Mozannar et al. (2024a)
78	Significant Productivity Gains through Programming with Large Language Models	Weber et al. (2024)
79	The RealHumanEval: Evaluating Large Language Models' Abilities to Support Programmers	Mozannar et al. (2024c)
80	The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers	Prather et al. (2024)
81	Toward Effective AI Support for Developers: A survey of desires and concerns	Khemka and Houck (2024)

Table 1 (continued)

ID	Title	Authors
82	Towards Feature Engineering with Human and AI's Knowledge: Understanding Data Science Practitioners' Perceptions in Human&AI-Assisted Feature Engineering Design	Zhu et al. (2024)
83	Transforming Software Development: Evaluating the Efficiency and Challenges of GitHub Copilot in Real-World Projects	Pandey et al. (2024)
84	Trust in Generative AI among students: An Exploratory Study	Amoozadeh et al. (2024)
85	Using AI Assistants in Software Development: A Qualitative Study on Security Practices and Concerns	Klemmer et al. (2024)
86	Validating AI-Generated Code with Live Programming	Ferdowsi et al. (2024)
87	When to Show a Suggestion? Integrating Human Feedback in AI-Assisted Programming	Mozannar et al. (2024b)
88	Exploring the Design Space of Cognitive Engagement Techniques with AI-Generated Code for Enhanced Learning	Kazemitabaar et al. (2025)
89	Exploring the problems, their causes and solutions of AI pair programming: A study on GitHub and Stack Overflow	Zhou et al. (2025b)
90	Generation Probabilities Are Not Enough: Uncertainty Highlighting in AI Code Completions	Vasconcelos et al. (2025)

**Fig. 1** Flow diagram of the study

- Limited number of educational studies highlighting the need for a deeper exploration of AI's impact on practical skills acquired during CS education, especially as developers constantly learn and re-learn programming practices in development environments.
- Lack of specification of the SDLC stage, suggesting an opportunity for future research to contextualize AI tool usage across all stages of the SDLC.

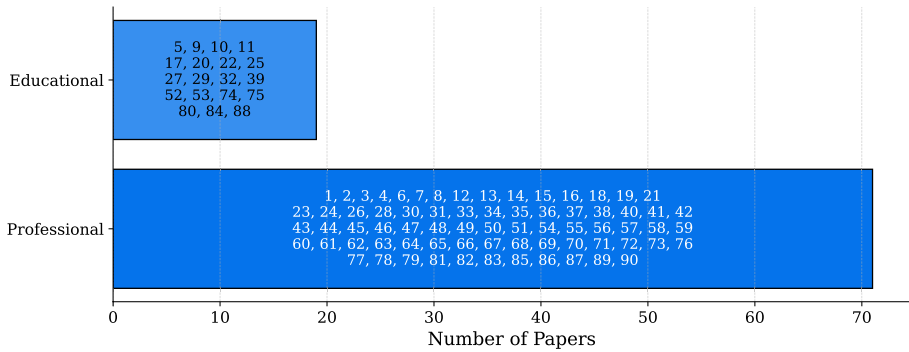


Fig. 2 Paper IDs by context

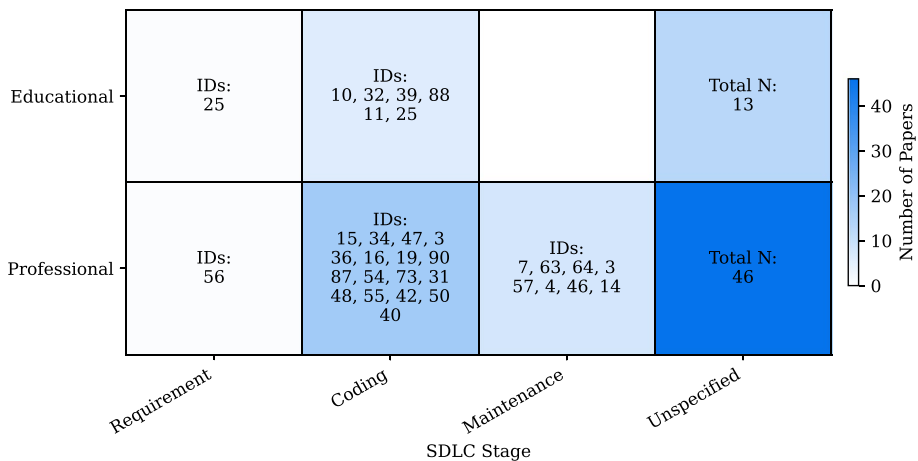


Fig. 3 Paper IDs by context and SDLC stage

Analysis of tasks assigned to the participants of studies reveals that they change over the three years. Early studies rely on smaller algorithmic exercises similar to LeetCode³-type programming tasks. Later studies introduce tasks that require data analysis, or a project-level use of AI. Data analysis tasks become more common over time. Studies add wrangling, plotting, and notebook workflows, which reflects the maturity of AI support in these environments and the relevance of studying such use. This broadening is also visible in the growth of the *Unspecified* category in Fig. 4, where authors evaluate AI assistance as general programming support without tying it to a single topic or SDLC stage. The variety of assigned tasks increases across years. The corpus now includes not only algorithmic coding and data analysis, but also documentation, testing, and a few cases of debugging and migration. This spread enables more precise claims about where AI helps, but also raises the need for clearer task descriptions to support comparison across studies.

A subset of studies conceptualizes programming with AI as a form of *pair programming*, where AI acts as a collaborative partner (Zhou et al. 2025b; Imai 2022; Zhou and Li

³LeetCode: a resource for technical interviews materials and coding competitions <https://leetcode.com/>

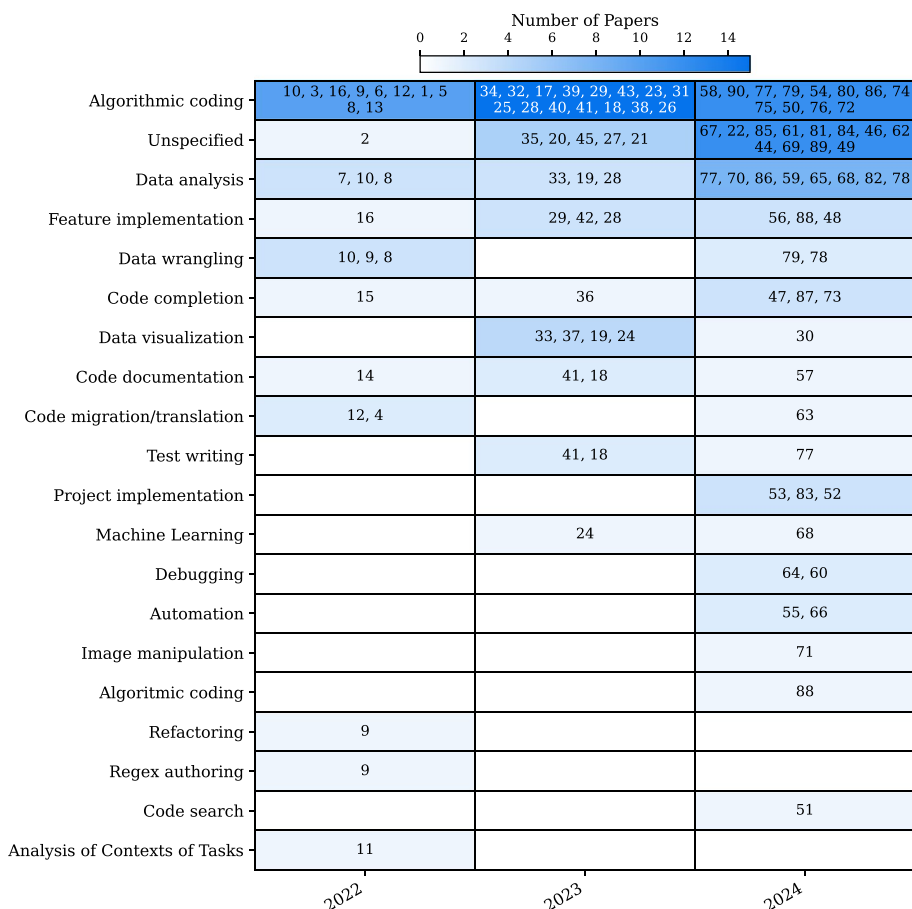


Fig. 4 Paper IDs by types of tasks and year

2023; Bird et al. 2022; Kazemitabaar et al. 2023a). This perspective highlights the evolving view of AI as not just a tool but a collaborator, representing a growing trend in co-creative programming workflows and suggesting new opportunities and challenges for both practitioners and educators.

3.2 Impact

Most prominent branch of research is dedicated to *Impact of HAX in IDE* with 74 out of 90 (see Fig. 5 for IDs) reviewed studies focusing on it. The increasing integration of AI-powered tools in IDEs has reshaped the way developers engage with coding, debugging, documentation, and software design (Peng et al. 2023; Weber et al. 2024). AI's role extends beyond simple automation, influencing productivity and software engineering practices, e.g., (Nguyen and Nadi 2022; Vaithilingam et al. 2022). Although AI has undoubtedly enhanced productivity in some contexts, its overall effects remain highly context-dependent

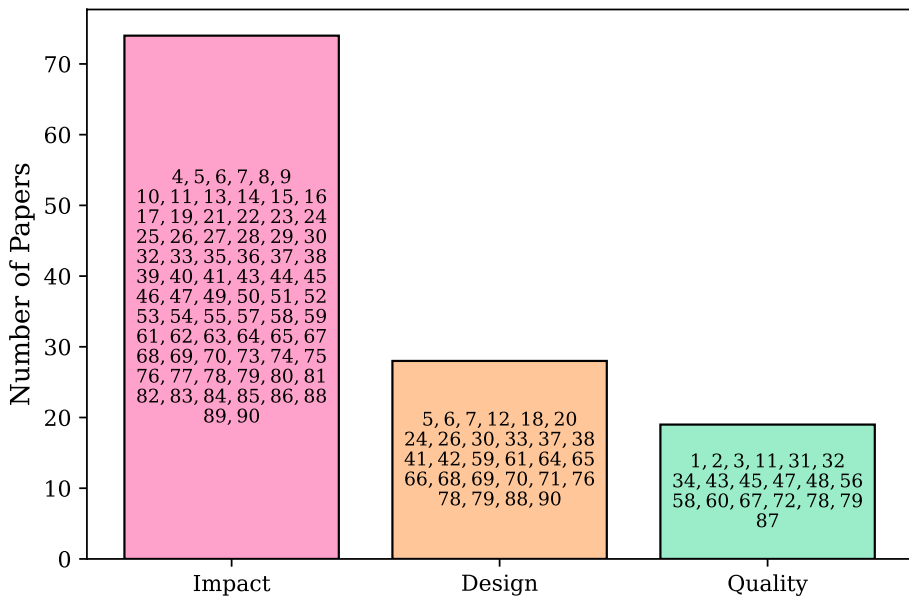


Fig. 5 Paper counts and IDs by scope

and nuanced. Studies consistently highlight the dual nature of AI’s impact, providing both benefits and challenges.

Multiple studies looked into the productivity aspect of coding with AI (13/74). Some of them have confirmed a notable boost in productivity when developers incorporate AI into their workflows. Potential benefits include a higher “ceiling” of productivity for experienced developers and faster onboarding for newcomers (Vasiliniuc and Groza 2023). For instance, participants using GitHub Copilot completed implementation of an HTTP server in JavaScript up to 55.8% faster than control groups (Peng et al. 2023), while other investigators observed productivity gains ranging from 26% to 35% for complex tasks, across many files, and in proprietary contexts (Pandey et al. 2024). These improvements are partially attributed to fewer context switches and reduced boilerplate coding, enabling developers to offload repetitive tasks to AI while focusing on higher-level logic. However, gains are inconsistent: developers report that tasks involving proprietary or highly complex logic see only marginal benefits, as current assistance struggles with context-specific nuances of the real-world programming (Khemka and Houck 2024).

Although AI-driven suggestions frequently accelerate development, they also require new verification efforts. When using AI tools, verifying suggestions, refining prompts, and reworking code generated by AI might take up to 50% of developers’ time (Mozannar et al. 2024a). This increased mental overhead arises because AI’s output can be partially correct yet subtly flawed (Wermelinger 2023), requiring careful review and, at times, stepwise interaction or re-prompting (Kazemitabaar et al. 2024a). Recent work proposes mechanisms that reduce this cost, showing that highlighting tokens by predicted edit likelihood improves speed and directs attention to problematic regions (Vasconcelos et al. 2025). Such extra validation often mitigates the risk of blindly accepting erroneous suggestions, *i.e.*, phenomenon known as “automation bias” (Al Madi 2022).

Studies focusing on novice programmers reveal both benefits and pronounced risks. Across classroom and lab settings, AI assistance tends to speed novice work and reduce perceived workload while short-term learning remains intact when use is structured (Kazemitabaar et al. 2023a). Beginners often solve programming assignment quicker with AI's help (Puryear and Sprint 2022; Rasnayaka et al. 2024; Tanay et al. 2024). Time-pressured studies report lower mental workload and steady self-efficacy with assistance, with additional time using AI relating to better task outcomes (Gardella et al. 2024). However, beginners can over-trust AI, losing sight of important foundational concepts (Prather et al. 2024). In certain cases, when system suggestions are misleading, novices drift away from correct solutions, as they lack the expertise to detect subtle errors (Zhou and Li 2023). Structured prompts and explanatory interfaces can help novices understand AI outputs better, fostering safer learning pathways (Kruse et al. 2024). A hybrid strategy that mixes authoring and modification is associated with better results in self-paced CS1 tasks, and prompting is a learnable activity that raises solve rates when taught explicitly (Kazemitabaar et al. 2023b; Denny et al. 2023).

3.2.1 Attitude

A subset of studies (11/74) examines user attitude toward AI assistance in IDEs, with trust as the central construct. Trust is shaped by both the properties of the suggestions and the expertise of the user (Amoozadeh et al. 2024; Brown et al. 2024). High-quality and context-relevant suggestions raise trust, while inconsistency and unnecessary complexity lower it. Layers of outputs that help users judge suggestions, such as brief explanations, usage statistics, and control over scope, are proposed as practical supports for trust formation (Wang et al. 2024). Context also matters. Trust drops for high-stakes or production-related work and for complex or open-ended tasks, and it rises for routine work or proof-of-concept code (Wang et al. 2024). In addition, developers treat suggestions in test files more cautiously than those in production files, and they are less likely to accept code completion suggestions when writing tests (Brown et al. 2024).

Attitude relates to reliance. Over-reliance appears when users accept AI output without sufficient checking, and under-reliance appears when they dismiss correct output because justification feels costly. Novices often overestimate the reliability of AI output and invest less time in verification for algorithmic problems (Al Madi 2022), so they tend to over-rely on AI. In educational settings, reliance and trust vary widely and depend on perceived performance of the system and task context, which warrants explicit trust calibration rather than blanket bans (Amoozadeh et al. 2024; Brown et al. 2024; Wang et al. 2024; Lau and Guo 2023). Students also renegotiate authorship and autonomy when assistants anticipate intent, which calls for assessment that values reasoning traces and verification (Prather et al. 2023). Moreover, novices struggle to form accurate mental models of synthesizers, so interfaces should expose system state and allow incremental control (Jayagopal et al. 2022). Professional developers sometimes under-rely and reject correct outputs that lack a clear rationale (Sun et al. 2022). Tooling can mediate both patterns. Live programming environments that surface runtime values reduce the effort of review and support more calibrated acceptance decisions (Ferdowsi et al. 2024).

3.3 Design

To account for the changes introduced by AI integration in IDEs, a focused body of research has developed on the *Design* of HAX in IDE. Among the reviewed studies, 28 out of 90 (see Fig. 5 for IDs) investigate forms of AI integration into IDEs, with most of them (22/28) explicitly examining how these forms of integration impact users, and 17 of them exploring design principles.

The research presents two primary paradigms of AI-assisted interaction inside IDEs: autocompletion-based assistance and conversational agents. Autocompletion interfaces provide short, low-friction code completions, enabling developers to work seamlessly with inline suggestions. These approaches minimize cognitive load and support rapid decision-making without disrupting workflow (Vaithilingam et al. 2023; Yen et al. 2023). In contrast, conversational interactions (often embodied in chat-based assistants or virtual avatars) facilitate higher-level reasoning, iterative refinement, and complex problem-solving (Ross et al. 2023a; Robe and Kuttal 2022; Gu et al. 2024). Rather than being opposing paradigms, these interaction modes are complementary (Weber et al. 2024), with studies advocating for hybrid models that seamlessly integrate autocompletion with interactive, context-sensitive conversations. In addition, some studies experiment with alternative modalities of interaction that go beyond these two dominant paradigms. For example, mixed-reality interfaces (Manfredi et al. 2023) introduce conversational AI in augmented pair-programming environments, leveraging immersive learning experiences.

Due to the conversational nature of interaction with AI in IDE, prompting became a prominent object of investigations with 7 out of 28 studies focusing on this aspect. Across reviewed studies, prompt engineering is positioned as a strategic skill that shapes how developers and AI systems collaborate, affecting both individual productivity and team-based workflows (Kruse et al. 2024; Yen et al. 2023). There are claims that prompting should be taught as a skill, with scaffolds for decomposition, short rationales before revealing solutions, and reusable prompt solution pairs delivered in step-by-step dialogues (Kazemitabaar et al. 2025). A central challenge in AI-assisted development is ensuring that AI-generated code aligns with user intent (Fagadau et al. 2024; Denny et al. 2023; OBrien et al. 2024). Therefore, studies investigating prompt formulation demonstrate that structured and well-contextualized prompts lead to more accurate and reliable output, while excessive detail can degrade performance. As AI tools become embedded in professional and educational workflows, collaborative prompt engineering is also gaining importance (Feng et al. 2024; Cheng et al. 2024b). Consequently, authors introduce frameworks for shared prompt refinement, knowledge transfer, and workflow optimization, suggesting the need for modularity and re-usability in AI interactions.

For education, in-IDE HAX should shift learners from code consumption to deliberate practice. Interfaces need small, purposeful friction before code adoption, which would improve transfer and calibration without added load (Kazemitabaar et al. 2025). Contribution of AI should be staged and gated by quick checks (Robe and Kuttal 2022; Kazemitabaar et al. 2025). Embodied variants can sustain engagement with the task when a studying partner is absent, as mixed reality pair-programming with a conversational avatar ties feedback to editor context (Manfredi et al. 2023). Finally, to reduce over-reliance, systems should make performance on similar tasks completed without assistance visible for self-assessment (Kazemitabaar et al. 2025).

Beyond interaction paradigms, 17 out of 28 studies explore design principles for AI-powered systems inside IDEs, for instance (Chen and Zacharias 2024; Robe and Kuttal 2022; Jayagopal et al. 2022; Cheng et al. 2024a; Yen et al. 2023; Shlomov et al. 2024; Guo et al. 2024; McNutt et al. 2023; Wang et al. 2023b; Feng et al. 2024; Gu et al. 2024; Angert et al. 2023; Kazemitabaar et al. 2024a). These principles guide how AI should be embedded into developer workflows and environments while maximizing usability, reliability, and efficiency:

- **Context Awareness:** LLM-powered assistance significantly boosts task completion rates when it can effectively incorporate a developer's code as context (Nam et al. 2023; Gu et al. 2024).
- **Explainability & Transparency:** Tools should provide context-aware explanations and highlight uncertain AI-generated code (Vasconcelos et al. 2025; Yan et al. 2024; Sun et al. 2022; Wang et al. 2024; Vaithilingam et al. 2023) to improve user trust.
- **User Control & Adaptability:** Users need adaptive, fine-tuned to expertise or business context AI responses, and scaffolded AI guidance to prevent over-reliance (Prather et al. 2024; Jiang et al. 2022; Cheng et al. 2024b).

Among implementations of AI for IDEs, GitHub Copilot is by far the most studied, with 36 out of 90 works investigating it. This concentrated attention on a single tool suggests a risk of overgeneralization, where findings may not fully translate to other AI-powered coding assistants, limiting broader design and implementation insights.

However, research also explores alternative AI-powered tools and approaches beyond Copilot, highlighting a growing landscape of intelligent development assistants. For example, IDA (Shlomov et al. 2024) introduces a no-code automation system tailored for non-programmers, while Prompt Sapper (Cheng et al. 2024b) supports AI chain development through modular and visual programming interfaces. Studies, such as CoLadder (Yen et al. 2023), investigate hierarchical prompt structures to refine interactions with code-generation models. PairBuddy (Robe and Kuttal 2022) explores conversational AI as a simulated pair-programming partner. Domain-specific solutions, such as Slide4N (Wang et al. 2023b) for human-AI collaboration in computational notebooks and RealHumanEval (Mozannar et al. 2024c) for LLM performance evaluation in software engineering, illustrate the diversity of AI integrations in development workflows.

3.4 Quality

Investigating and ensuring the *Quality* of AI-assisted code generation is a central concern of 19 out of 90 papers in the reviewed literature (see Fig. 5 for IDs). Across studies, quality is examined from multiple perspectives, including code correctness, security, and readability.

One recurring theme is the trade-off between efficiency and correctness. AI-assisted coding significantly accelerates development workflows, but at the cost of increased susceptibility to subtle errors, security vulnerabilities, and reduced maintainability (Sandoval et al. 2023; Tang et al. 2024; Pearce et al. 2022). Several studies suggest that while AI-generated code often appears syntactically valid, it can contain logical flaws, insecure patterns, or deviations from best practices (Asare et al. 2024; Pearce et al. 2022). As a result, effective quality control mechanisms, such as automated verification, static analysis, and model-

aware debugging tools, are emphasized as critical to responsible adoption (Agarwal et al. 2024).

Beyond correctness, readability and maintainability emerge as key aspects of AI-generated code quality. Although AI can generate complex solutions rapidly, its output is not always optimized for human understanding. Studies highlight that AI-generated code may be less readable and comprehensible due to overly concise structures, unconventional variable naming, or lack of meaningful comments (Al Madi 2022). This raises concerns about long-term maintainability, particularly in team-based software development workflows, where code legibility is as important as functional correctness. Research suggests that integrating AI-powered refactoring and explanation tools can help mitigate these challenges, enhancing both short-term efficiency and long-term code quality (Yan et al. 2024).

Studies additionally pose security as an additional concern: up to 36% of vulnerabilities in AI-assisted code originate from the LLMs, pointing out the risks of replicating insecure patterns from training data (Sandoval et al. 2023), emphasizing the need for cautious oversight of AI suggestions.

Overall, while LLM-powered tools can improve the software development process, maintaining consistently high-quality code remains a challenge. Ensuring reliability requires better alignment with developers' needs and robust verification mechanisms (Tang et al. 2024; Agarwal et al. 2024). Studies highlight the need for strategies such as filtering suboptimal suggestions, improving contextual prompting through docstrings and function names (Mozannar et al. 2024c; de Moor et al. 2024). Moreover, research calls for adaptive personalisation of code suggestions to support the overall quality of AI-generated code. In practice, this means learning per-developer and per-task policies for when to suggest, which modality to use, how much code to propose, and how much context to include in the prompt (Mozannar et al. 2024c; de Moor et al. 2024).

Quality of code in AI-assisted development is not just a technical issue but also a human and organizational challenge. Research highlights that quality outcomes are influenced by how developers interact with AI tools: whether they verify AI-generated code, how they prompt AI models, and whether they develop strategies for AI-assisted debugging (Weber et al. 2024; Mozannar et al. 2024c; Sandoval et al. 2023; Al Madi 2022; de Moor et al. 2024). Best practices for improving AI-assisted development quality extend beyond tool design and include education, training, and overall workflow adaptation.

3.5 Methodologies

There are multiple methodological approaches used in human-AI interaction research. These methodologies can broadly be categorized into *qualitative* and *quantitative* approaches. Quantitative methods, in turn, can be further divided into survey-based studies and experimental research designs. Among 90 papers reviewed in this study, 12 employed a survey methodology, 32 followed an experimental design, and 38 utilized qualitative methods.

Qualitative studies included a range of approaches, from semi-structured interviews to more complex methods such as the Wizard of Oz paradigm (Robe and Kuttal 2022), focus group interviews (Vaithilingam et al. 2023), and Grounded Theory-based analysis (Li et al. 2024). Some studies adopted a mixed-methods approach, combining multiple methodologies. For instance, an experimental design complemented by participant interviews (a format often referred to as a user study).

Additionally, 20 of the reviewed studies did not fit neatly into the qualitative-quantitative dichotomy. Examples include case studies in which the authors both describe their own experiences and provide quantitative metrics on system behavior (Ross et al. 2023b), quantitative analysis of system behavior using pre-defined datasets of typical tasks (Nguyen and Nadi 2022; Fagadau et al. 2024), and investigations of public discussions on platforms such as Reddit (Klemmer et al. 2024) and GitHub Issues (Zhou et al. 2025b).

The sample sizes for different methodological approaches varied considerably. For interview-based studies, the sample size ranged from 7 to 61 participants (median = 16). Experimental designs involved sample sizes ranging from 17 to 214 participants, with the exception of two A/B studies, one with 535 participants (Mozannar et al. 2024b) and another, conducted by Meta (Dunay et al. 2024), reporting “thousands” of participants. As expected, survey-based studies had the largest sample sizes, ranging from 68 to 2,047 participants (median = 507).

3.6 Future Work Suggested by Existing Literature

To address RQ3, we analyzed 250 future work statements extracted from the reviewed papers (see the full table in Sergeyuk et al. 2025). Each item was open-coded by topic (what is proposed) and method (how it is to be studied), resulting in 17 topics and 10 methodological strategies.

In Fig. 6, we present corresponding Paper IDs for each topic, and Fig. 7 visualizes the distribution of papers across combinations between topics and methods.

By analyzing the future work proposed by each study alongside the goals of others, we identified alignment patterns and gaps that highlight both the progress made and the opportunities missed. Some future work suggestions align with the goals of other studies, but these connections are not always explicit, reflecting a fragmented research landscape. Common topics like productivity, audit of AI-generated code, and usability of designed AI tools appear across studies but are addressed in isolated contexts without direct cross-referencing. This lack of continuity limits the field’s ability to build cumulative knowledge effectively.

Suggestions cluster in a set of 17 themes. The most common address productivity factors (43), the design of AI assistance (29), and audits of AI-generated code (28). Additional areas include prompting support (19), skill building with AI (17), education with AI (16), personalization (16), explainability and transparency (15), verification support (13), and broader coverage of the SDLC (12). Less frequent but present are trust and reliability (11), IDE redesign for AI (10), assistant context enrichment (6), user mental models (5), proactivity (4), user control (3), and AI governance (3).

Productivity-oriented suggestions emphasize broader sampling and comparative designs, often at a larger scale (Al Madi 2022; Prather et al. 2023; Angert et al. 2023; Jiang and Coblenz 2024; Rasnayaka et al. 2024). They ask for studies with professional developers in realistic settings (Peng et al. 2023; Li et al. 2024; Jiang and Coblenz 2024; Rasnayaka et al. 2024; Zhu et al. 2024; Kazemitabaar et al. 2025) and for attention to students and novices (Jiang and Coblenz 2024; Amoozadeh et al. 2024; Kazemitabaar et al. 2025). Authors also recommend evaluations that account for project and environment factors (Peng et al. 2023; Feldman and Anderson 2024; Zhou et al. 2025b). Several papers connect productivity to interface and workflow choices and propose concrete design changes and con-

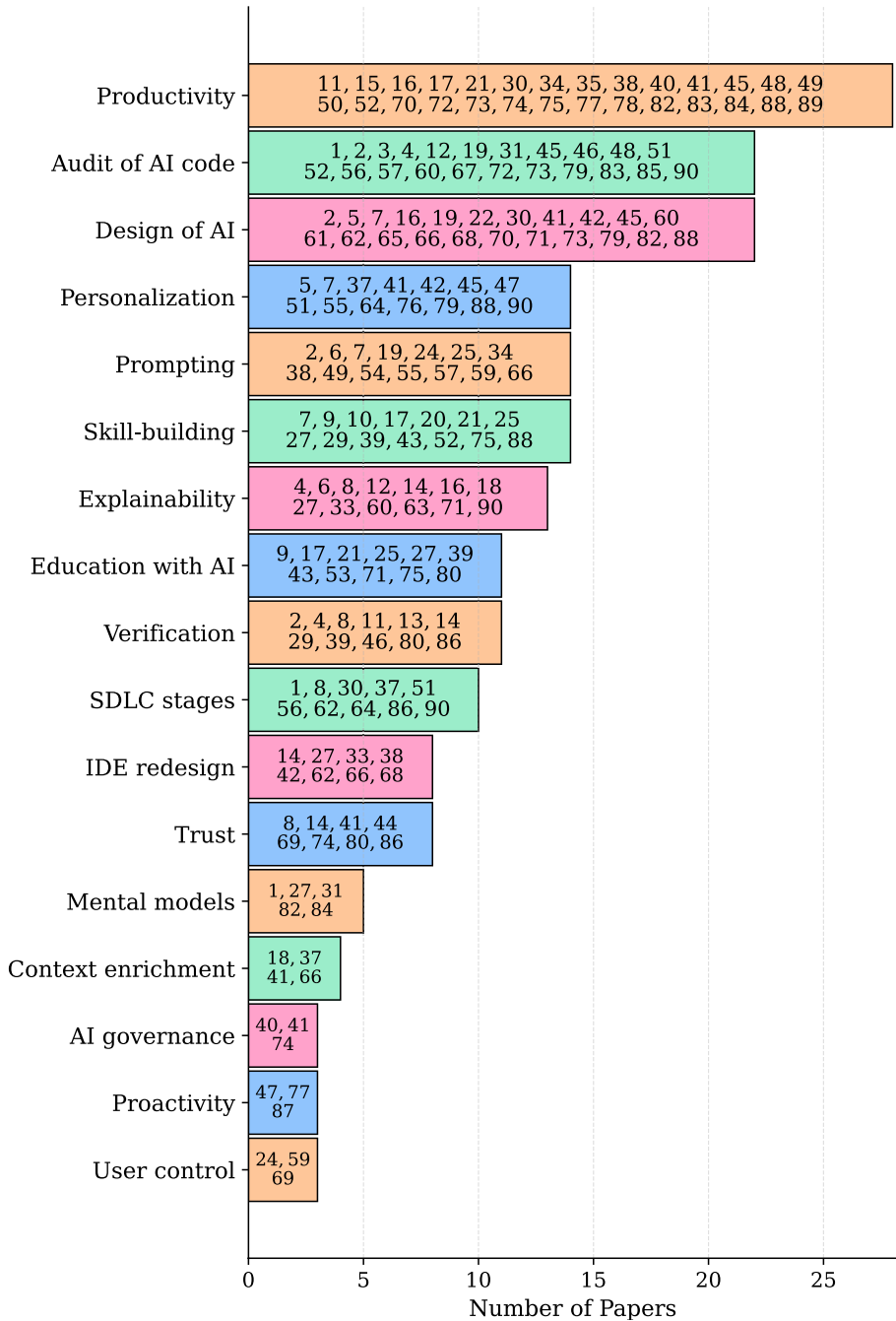


Fig. 6 Paper IDs for topics of suggested future work

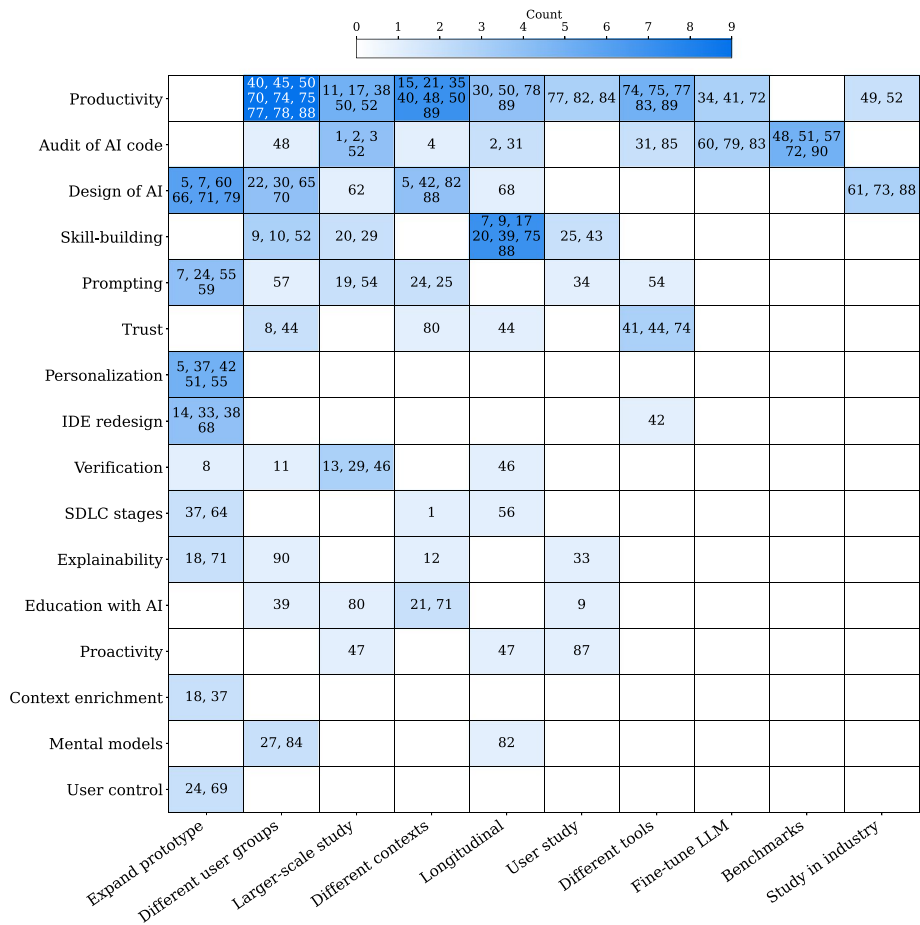


Fig. 7 Paper IDs for co-occurrence between TOPIC on the y-axis and METHOD on the x-axis for suggested future work

trols (Ziegler et al. 2022; Angert et al. 2023; Rasnayaka et al. 2024; Ságodi et al. 2024; Pandey et al. 2024; Amoozadeh et al. 2024; Nam et al. 2023; Jiang and Coblenz 2024).

Future work on auditing AI-generated code seeks wider security coverage, longitudinal observation, and stronger assets for evaluation. Proposals include expanding vulnerability categories and language coverage and tracking change over time (Pearce et al. 2022; Asare et al. 2023, 2024), building targeted test suites and benchmarks (Liu et al. 2024), and increasing unit tests and quality metrics to stress code quality (Yetistiren et al. 2022; Liu et al. 2024). Some papers extend audits to SDLC touchpoints and to interfaces that surface risk at decision time (Liang et al. 2024; Asare et al. 2024; Liu et al. 2024; Kruse et al. 2024; Pearce et al. 2022; Vasconcelos et al. 2025).

Prompting support centers on guided interactions and safe practice. Suggestions include structured conversational prompting and refinement aids (Jiang et al. 2022; Angert et al. 2023), prompt engineering strategies with comparative evaluation (Pearce et al. 2022; Denny et al. 2023; Mastropaolo et al. 2023; Fagadau et al. 2024), multi-modal prompt chan-

nels for tasks that are not purely textual (Angert et al. 2023), and prompt sanitization with privacy-aware feedback (Li et al. 2024). Authors also ask for generalization across tools and settings (Fagadau et al. 2024).

Personalization proposals focus on adapting assistance to user profiles, expertise, and task state. Examples include user models to personalize initial prompts and responses (Ross et al. 2023a; Chopra et al. 2024), frequency and snooze controls to manage attention (Vaithilingam et al. 2023), adaptive strategies that respond to real time performance and needs (Kazemitabaar et al. 2025), and results that better match the intended coding context and style (de Moor et al. 2024; Liu et al. 2024; Sahoo et al. 2024; Cheng et al. 2024b; Mozannar et al. 2024c; Vasconcelos et al. 2025). Several papers request tailored documentation and scaffolds for specific roles such as data analysts (Wang et al. 2022; Gu et al. 2024).

Explainability and transparency aim to help users understand, validate, and teach the assistant. Suggested mechanisms include rationales, annotations, and code level explanations (Vaithilingam et al. 2022; Hu et al. 2022; McNutt et al. 2023), interpretability features that support accurate mental models and calibrated expectations (Weisz et al. 2022; Jiang et al. 2022; Sun et al. 2022; Bird et al. 2022), internal deliberation to improve reasoning quality (Ross et al. 2023b), and presentation of validation evidence such as testing suite results or multi model checks (Omidvar Tehrani et al. 2024; Yan et al. 2024).

Verification support concentrates on tools that help users check and repair outputs before adoption. Proposals include automatic test generation and retrieval of reference examples (Vaithilingam et al. 2022; Ferdowsi et al. 2024), post processing and filtering layers that repair or block risky outputs (Pearce et al. 2022; Al Madi 2022; Ferdowsi et al. 2024), interface patterns that present multiple alternatives for comparison (Weisz et al. 2022), and study designs that measure how learners verify AI suggestions and develop durable checking habits (Kazemitabaar et al. 2023b, a).

Coverage beyond the implementation stage remains a consistent request. Papers propose research and tooling for requirements, design, evolution, debugging, testing, deployment, and code search across the life cycle (Nguyen and Nadi 2022; Liu et al. 2024; O'Brien et al. 2024; Chopra et al. 2024; Ferdowsi et al. 2024; Vasconcelos et al. 2025). Many of these suggestions pair life cycle scope with deeper integration into the developer workspace (Chopra et al. 2024; Ferdowsi et al. 2024).

Trust and reliability suggestions examine individual and community factors as well as interface effects. Authors ask for studies of how communities and organizations form and recalibrate trust over time (Cheng et al. 2024a; Prather et al. 2024), for quantification of interface-driven trust shifts (Wang et al. 2024), for evaluations that measure the impact of assistant behavior changes on acceptance (Feldman and Anderson 2024), and for approaches that foster appropriate trust with clear boundaries (Ferdowsi et al. 2024). Inclusion of diverse developer groups appears throughout (Vaithilingam et al. 2022; Hu et al. 2022; Ross et al. 2023a).

Proposals on IDE redesign and context enrichment argue for AI-aware environments. Suggestions include non-linear input and new interaction media (McNutt et al. 2023), improved error handling and better merging of sketch-like artifacts (Angert et al. 2023), chat or dialogue interfaces for automation control (Shlomov et al. 2024), and consistent propagation of edits and assumptions (Kazemitabaar et al. 2024a). For context, papers recommend memory management, dynamic prompts that reflect project artifacts, and search-based integrations (Ross et al. 2023b; Wang et al. 2023b; Ross et al. 2023a; Shlomov et al. 2024).

Less frequent but important themes include user mental models, proactivity, user control, and governance. Future work asks for theory building on how developers form and use mental models of assistants and for longitudinal observation of expectation change (Nguyen and Nadi 2022; Lau and Guo 2023; Asare et al. 2023; Zhu et al. 2024; Amoozadeh et al. 2024). Proactivity-related suggestions seek predictive models of user state and careful study of long-term effects on quality of code and productivity (de Moor et al. 2024; Mozannar et al. 2024a, b). Use control-oriented suggestions call for clearer and richer controls over generation and automation levels (Yen et al. 2023; Feng et al. 2024; Wang et al. 2024). Governance suggestions raise questions about fairness, access, and alignment, including alignment toward helpful and harmless behavior and possible career impacts for different demographic groups (Ross et al. 2023a; Feldman and Anderson 2024; Peng et al. 2023).

Taken together, these directions indicate the need for larger and longer evaluations, stronger audit and verification assets, broader life cycle coverage, and adaptive assistance that is explainable and under user control.

4 Discussion

We organize the discussion of our systematic literature review of 90 in-IDE HAX studies around topics coverage and gaps (RQ1), implications for practice and research (RQ2), and a future work agenda (RQ3).

RQ1: Extensively studied and under-explored aspects of in-IDE HAX The corpus of reviewed works separates cleanly into two contexts of use for AI in software development, namely professional (71/90) and educational (19/90). Within professional settings, most papers examine practical use in coding and maintenance, and a large share evaluate assistance without situating it in a specific SDLC stage. Educational papers focus on how learners use assistance to solve programming problems. Here as well, stage information is often omitted, which limits comparison across contexts. This weakens interpretability because the same interface may have different effects at requirements, implementation, testing, or evolution. Future work should report the stage explicitly and use a simple task taxonomy that allows cross-paper synthesis. Task design employed in studies evolves over time. Early studies rely on small algorithmic exercises that resemble simple games and interview-style problems. More recent work introduces project-level analysis of workflows and begins to include tasks related to documentation, testing, debugging, and migration. This broadening improves ecological validity, yet it also increases the number of papers that label the task as unspecified. Clearer task descriptions and stable taxonomies are needed to preserve comparability as the scope widens.

Several papers frame AI-assistance as a form of pair programming in which the system acts as a collaborator rather than a tool. This view aligns with the observed move from isolated completions to co-creative workflows. It also raises concrete needs in both contexts, namely training in model interaction and oversight in professional teams, and scaffolds that keep learners in control in educational settings. In summary, the evidence is richest for professional coding and maintenance, with a growing variety in task types and a notable reporting gap on the SDLC stage. The next step is to include non-adopters, expand cover-

age to early and late stages, and standardize task and stage reporting so that results can be compared across settings and over time.

RQ2: Key findings and implications in the field of HAX in IDEs *Impact.* Across reviewed studies, there is evidence that AI assistance in IDEs changes how developers create and verify code. The main benefit is faster progress on routine or well-scoped work, often explained by fewer context switches and reduced need for boilerplate code writing. The main limit appears on tasks that embed proprietary knowledge or complex logic, where current systems struggle to capture project-specific nuance (Peng et al. 2023; Pandey et al. 2024; Khemka and Houck 2024). Code verification is, therefore, central. About fifty percent of developers' time is now spent on inspecting suggestions, refining prompts, and repairing code (Mozannar et al. 2024a). These findings argue for budgeting verification explicitly and for evaluating impact with edits and verification effort in addition to the suggestions' acceptance rate and time spent on task. Effects are heterogeneous across populations. Novices complete tasks more quickly and report lower workload when the use of in-IDE AI is structured, yet they are vulnerable to subtle errors and may overtrust suggestions (Kazemitabaar et al. 2023a; Prather et al. 2024; Zhou and Li 2023). Professionals benefit from speed on familiar patterns but may under-rely on AI and reject correct suggestions if it lacks a clear rationale (Sun et al. 2022). Trust also varies by context. It decreases in high-stakes and testing contexts and increases for routine or proof-of-concept work, which argues for visible boundaries and rationale at decision time (Wang et al. 2024; Brown et al. 2024).

Design. Reviewed studies converge on three core design principles for AI in IDEs, namely rich context awareness, explainability with transparency, and user control with adaptability. These principles guide how AI systems should be built. Moreover, it is highlighted that the two dominant interaction modes, autocompletion and conversational, should work together. A smooth switch between them, a shared context pool, and consistent state across views allow developers to move from local edits to higher-level reasoning without friction (Weber et al. 2024; Yen et al. 2023). As conversational interaction with AI enters IDEs, prompting becomes an important learnable activity. Structured prompts and reusable prompt-outcome pairs align generation with intent and support calibrated acceptance (Kruse et al. 2024; Denny et al. 2023; Fagadau et al. 2024). Interfaces that ask the user to state the next step before code is shown and that supply brief rationales further stabilize decisions (Kazemitabaar et al. 2025), as well as step-by-step explanations of the generated code for code validation and refinement (Tian et al. 2023, 2024). Explanation and control must be available at decision time. Useful aids in verification include surfacing runtime values, side-by-side alternatives, retrieval of reference examples, static analysis, and unit tests inside the editor (Ferdowski et al. 2024; Pearce et al. 2022; Weisz et al. 2022; Vaithilingam et al. 2022). Context-aware explanations and explicit scope controls make it clear where and why the assistant acts (Vaithilingam et al. 2023; Wang et al. 2024; Yan et al. 2024). When available, calibrated uncertainty cues or predicted edit likelihood should direct review effort, without relying on raw generation probabilities alone (Spiess et al. 2025; Vasconcelos et al. 2025). In educational settings, structured prompts, brief rationales, staged contribution, and quick checks inside the IDE help learners stay in control (Robe and Kuttal 2022; Kazemitabaar et al. 2024b, 2023a; Prather et al. 2024; Zhou and Li 2023). Embodied or mixed reality

agents are useful when the goal is to practice collaboration routines with feedback tied to the editor context (Manfredi et al. 2023).

Quality. Studies in this direction examine correctness, security, readability, and maintainability of code produced with AI assistance inside the IDE. Across these papers, the central pattern is a trade-off between speed and assurance. Assistance can accelerate progress, yet LLM-generated code that passes superficial checks may still contain logical flaws or insecure patterns, which shifts effort from authoring toward checking (Wang et al. 2025; Nguyen and Nadi 2022; Yetistiren et al. 2022; Al Madi 2022; Asare et al. 2023; Sandoval et al. 2023; Asare et al. 2024; Mozannar et al. 2024a; Izadi et al. 2024). LLM as judge signals can complement human review for correctness (Tong and Zhang 2024), but correctness outcomes depend not only on model properties but also on developer practice, including how prompts are written, how suggestions are inspected, and how teams organize review (Kruse et al. 2024; Feng et al. 2024; Kuang et al. 2024; Mozannar et al. 2024a). Readability and maintainability concerns recur alongside correctness. Short and compact suggestions reduce typing but can hinder clarity of intent through unconventional names, missing comments, or terse structure, which complicates later edits and team review (Nguyen and Nadi 2022; Yetistiren et al. 2022; Vaithilingam et al. 2022; Al Madi 2022). Explanatory aids and refactoring support inside the editor help address these issues and make suggested changes easier to justify (Yan et al. 2024; Ferdowsi et al. 2024). Security is a shared responsibility as well. Models can reproduce unsafe templates from training data, while developers may accept risky code under time pressure (Pearce et al. 2022; Sandoval et al. 2023; Asare et al. 2023, 2024; Al Madi 2022; Mozannar et al. 2024a; Vaithilingam et al. 2022; Gardella et al. 2024). Taken together, these findings call again for verification with tests and static analysis inside the editor, retrieval of relevant examples, and concise explanations at decision time, so that speed and assurance can coexist (Vaithilingam et al. 2022; Liu et al. 2024; Yan et al. 2024; Ferdowsi et al. 2024).

RQ3: Future research and development agenda Analysis of the future work statements yields a ranked set of topics and methodologies to study in-IDE HAX. The most frequent cluster concerns productivity factors (43). Here, the field should move from isolated gains to comparative and larger-scale evaluations that report acceptance, edits, and verification effort in addition to time. The next clusters focus on the design of assistance (29) and audits of AI generated code (28). For design, work should advance hybrid workflows that integrate autocompletion and conversation, with prompt scaffolds and explanation layers tied to decision points (Yen et al. 2023; Ross et al. 2023a; Brown et al. 2024; Guo et al. 2024). For audit, the field needs targeted test suites, stronger security coverage, and longitudinal assets that track change over time (Agarwal et al. 2024; Mozannar et al. 2024c; Asare et al. 2024; Pearce et al. 2022; Asare et al. 2023; Ságodi et al. 2024). Prompting support appears in nineteen statements and should prioritize guided decomposition and hierarchical specification, with interactive explanations that support validation and refinement (Yen et al. 2023; Brown et al. 2024). Skill building with AI (17) and education with AI (16) call for structured practice that protects learning goals inside authentic IDE settings (Prather et al. 2023; Zhou and Li 2023; Angert et al. 2023; Gardella et al. 2024; Ferdowsi et al. 2024). Personalization is equally frequent (16) and should aim at policies that adapt timing, modality, scope, and context to user profile and task state, with simple controls for frequency and snooze (Vaithilingam et al. 2023; Ross et al. 2023a; de Moor et al. 2024; Chopra et al.

2024; Koohestani and Izadi 2025). Explainability and transparency are requested by fifteen papers and should be evaluated for their effect on acceptance. Verification support (13) and broader SDLC coverage (12) align with the observed gaps in stage specification and with the shift of time toward checking. Less frequent but still important topics for study are trust and reliability (11), IDE redesign for AI (10), assistant context enrichment (6), user mental models, proactivity (4), user control (3), and governance (3). These numbers do not imply that lower frequency topics are unimportant. They indicate where evidence and assets are thin and where careful study can yield high value.

Methodological and open science recommendations Across the corpus, sample sizes are often underpowered and rarely justified. The median is seventeen participants. According to prior studies, for HCI practice, the most common sample size is twelve and the median is eighteen; in-person studies have a median of fifteen, while remote studies have a median of seventy-seven (Caine 2016). Although many quantitative studies at these sizes will be underpowered and therefore benefit from replication. We recommend a priori power analysis for confirmatory comparisons and saturation arguments for qualitative designs, with explicit reporting of recruitment constraints. Collaboration between academia and industry can improve recruitment and enable longitudinal assets. To support replication, we recommend sharing versioned repositories with study materials and data where feasible, including codebooks, prompts, task descriptions, analysis scripts, and deidentified datasets with clear metadata and a data dictionary. We also recommend using preregistration, blinding, and randomization when appropriate, and embedding transparency checklists that record methodology, exclusion criteria, and data sharing plans.

Takeaways Treat verification as a main part of the interaction with AI assistants, keeping tests, static analysis, and reference examples inside the editor and by surfacing runtime evidence to support calibrated acceptance decisions. Evaluate the effect beyond time spent on the task by reporting acceptance, edits, and verification effort. Design AI assistance as a hybrid of autocompletion and conversation that shares context, offers brief explanations, and scope controls at decision time. Structure educational use of AI in IDEs with staged contribution of AI, stepwise reveal of its suggestions, and controls for their quick checks. For studies, report the SDLC stage and expand coverage of tasks completed with AI to early and late stages. Strengthen methodological rigor through justified sample sizes, preregistration where appropriate, transparent reporting, and collaboration with industry for larger, ecologically valid, and longitudinal evaluations.

4.1 Threats to the Validity

While gathered findings provide valuable insights, in any scientific study, it is essential to consider potential threats to validity, which are factors that can affect the accuracy and generalizability of the research findings.

Sampling Bias Despite efforts to include well-known libraries and refine the search string, the possibility of sampling bias remains. Our recall-oriented query design can bias the pool toward professional contexts, since educational deployments do not always occur inside full IDEs or are indexed under alternative platforms. We mitigated this by including multiple

synonyms where supported and by manually screening every record for the inclusion criterion. Although our approach aimed to minimize bias, the inherent challenge of capturing every relevant work persists. That is why we provide our search protocol to mitigate this threat.

Temporal Bias The chosen time frame (papers published between 2022 and 2024) introduces a potential temporal bias, excluding earlier works. This decision was driven by the intention to focus on contemporary developments following the advent of LLMs. While acknowledging potential temporal limitations, we aim to capture the latest advancements and trends in this rapidly evolving field.

Source Reliability Inclusion of non-peer-reviewed papers from ArXiv introduces concerns regarding the reliability of findings, given the absence of a formal peer-review process. Recognizing this limitation, we deemed it necessary to consider insights from ArXiv due to the dynamic and rapidly evolving nature of the field. Thoroughly examining titles, abstracts, and full texts ensured that all included works contributed meaningfully to our survey. Moreover, the open publication environment of ArXiv encourages the publication of negative or null results, contributing to a more balanced representation of the research outcomes.

Interpretation Bias Analyzing a large amount of information can introduce interpretation bias and impact the way studies are categorized. While acknowledging this complexity, we emphasize transparency and provide the entire dataset (Sergeyuk et al. 2025) for the readers.

Challenges of industry-related research It is important to acknowledge that, overall, in-IDE HAX research faces unique challenges due to its close ties with the industry and professional context. One of the most significant challenges of conducting empirical studies in industrial settings is the "time factor". Industrial sponsors and participants often have short time horizons. They expect valuable feedback within 1-2 months of research, so that they could make timely business decisions. Although industrial sponsors and project managers are interested in rigorous results, practical constraints sometimes make it challenging to prioritize statistical significance. Analyzing immediate trends and patterns can be sufficient for them to assess the performance of their projects, teams, or technologies. By fostering a collaborative environment, we can develop strategies that balance the demands of industrial collaboration with the standards of academic research.

5 Conclusion

This review offers a structured synthesis of current research on AI-powered assistance in integrated development environments. Drawing on 90 studies, we identified dominant trends, conceptual gaps, and methodological characteristics of the emerging field of in-IDE HAX.

We found that research to date focuses heavily on a small set of tools, most often GitHub Copilot, and concentrates primarily on the implementation stage of the SDLC. While this has produced early insights into how developers engage with AI assistance during code writing and modification, it leaves other critical stages such as requirements analysis, test-

ing, and deployment underexplored. To make progress, the field must broaden its lens. Future research should investigate diverse AI assistants across varied development environments, extend coverage to earlier and later lifecycle stages, and prioritize longitudinal and comparative designs that capture how practices and their impact evolve over time. Attention is also needed to support personalization and trust calibration, reduce validation burden, and address broader quality and governance concerns. Finally, we note that methodological rigor remains an area for improvement. Methodologically, studies tend to prioritize short-term evaluations, often with underpowered samples and limited contextual variation. As the field matures, advancing cumulative evidence will require the adoption of robust empirical practices and closer collaboration across research groups.

In sum, the reviewed literature demonstrates growing interest in the integration of AI into developer tools but also highlights the need for deeper empirical grounding, broader tool coverage, and stronger methodological foundations. Addressing these challenges will be essential for supporting effective, transparent, and user-centered AI assistance in software development workflows.

Supplementary Information The online version contains supplementary material available at <https://doi.org/10.1007/s10664-025-10793-0>.

Acknowledgements This work was conducted as part of the AI for Software Engineering (AI4SE) collaboration between JetBrains and Delft University of Technology. The authors gratefully acknowledge the financial support provided by JetBrains, which made this research possible.

Author Contributions Agnia Sergeyuk, Ilya Zakharov, Ekaterina Koshchenko: Conceptualization, methodology, literature survey, writing, original draft, and editing. Maliheh Izadi: Conceptualization, methodology, writing, original draft, and review.

Funding This research was supported by JetBrains Research.

Data Availability All data are available in our materials (Sergeyuk et al. 2025).

Declarations

Ethical Approval Not applicable.

Informed consent Not applicable.

Conflict of interest The authors declare that they have no conflict of interest.

Clinical Trial Number Not applicable.

References

- Agarwal A, Chan A, Chandel S, Jang J, Miller S, Moghaddam RZ, Mohylevskyy Y, Sundaresan N, Tufano M (2024) Copilot evaluation harness: evaluating llm-guided software programming. [arXiv:2402.14261](https://arxiv.org/abs/2402.14261)
- Agbo FJ, Olivia C, Oguibe G, Sanusi IT, Sani G (2025) Computing education using generative artificial intelligence tools: a systematic literature review. *Computers and Education Open*, 100266
- Al Madi N (2022) How readable is model-generated code? examining readability and visual inspection of github copilot. In: *Proceedings of the 37th IEEE/ACM international conference on automated software engineering*, pp 1–5

- Alshamrani A, Bahattab A (2015) A comparison between three sdlc models waterfall model, spiral model, and incremental/iterative model. *Int J Comput Sci Issues (IJCSI)* 12(1):106
- Amoozadeh M, Daniels D, Nam D, Kumar A, Chen S, Hilton M, Srinivasa Ragavan S, Alipour MA (2024) Trust in generative ai among students: an exploratory study. In: *Proceedings of the 55th ACM technical symposium on computer science education V. 1*, pp 67–73
- Angert T, Suzara M, Han J, Pondoc C, Subramonyam H (2023) Spellburst: a node-based interface for exploratory creative coding with natural language prompts. In: *Proceedings of the 36th annual ACM symposium on user interface software and technology*, pp 1–22
- Ariza JÁ, Restrepo MB, Hernández CH (2025) Generative ai in engineering and computing education: a scoping review of empirical studies and educational practices. *IEEE Access*
- Asare O, Nagappan M, Asokan N (2023) Is github's copilot as bad as humans at introducing vulnerabilities in code? *Empir Softw Eng* 28(6):129
- Asare O, Nagappan M, Asokan N (2024) A user-centered security evaluation of copilot. In: *Proceedings of the IEEE/ACM 46th international conference on software engineering*, pp 1–11
- Barke S, James MB, Polikarpova N (2023) Grounded copilot: how programmers interact with code-generating models. *Proc ACM Program Lang* 7(OOPSLA1):85–111
- Bird C, Ford D, Zimmermann T, Forsgren N, Kalliamvakou E, Lowdermilk T, Gazit I (2022) Taking flight with copilot: early insights and opportunities of ai-powered pair-programming tools. *Queue* 20(6):35–57
- Brown A, D'Angelo S, Murillo A, Jaspan C, Green C (2024) Identifying the factors that influence trust in ai code completion. In: *Proceedings of the 1st ACM international conference on AI-powered software*, pp 1–9
- Caine K (2016) Local standards for sample size at chi. In: *Proceedings of the 2016 CHI conference on human factors in computing systems*, pp 981–992
- Carrera-Rivera A, Ochoa W, Larrinaga F, Lasa G (2022) How-to conduct a systematic literature review: a quick guide for computer science research. *MethodsX* 9:101895
- Centre for Reviews and Dissemination (UK) (1995) Database of abstracts of reviews of effects (dare): quality-assessed reviews. <https://www.ncbi.nlm.nih.gov/books/NBK285222/>. Accessed 10 Aug 2025
- Chen J, Zacharias J (2024) Design principles for collaborative generative ai systems in software development. In: *International conference on design science research in information systems and technology*. Springer, pp 341–354
- Cheng R, Wang R, Zimmermann T, Ford D (2024a) “it would work for me too”: how online communities shape software developers' trust in ai-powered code generation tools. *ACM Trans Interact Intell Syst* 14(2):1–39
- Cheng Y, Chen J, Huang Q, Xing Z, Xu X, Lu Q (2024b) Prompt sapper: a llm-empowered production tool for building ai chains. *ACM Trans Softw Eng Methodology* 33(5):1–24
- Chopra B, Bajpai Y, Biyani P, Soares G, Radhakrishna A, Parnin C, Gulwani S (2024) Exploring interaction patterns for debugging: enhancing conversational capabilities of ai-assistants. [arXiv:2402.06229](https://arxiv.org/abs/2402.06229)
- Denny P, Kumar V, Giacaman N (2023) Conversing with copilot: exploring prompt engineering for solving cs1 problems using natural language. In: *Proceedings of the 54th ACM technical symposium on computer science education V. 1*, pp 1136–1142
- Dunay O, Cheng D, Tait A, Thakkar P, Rigby PC, Chiu A, Ahmad I, Ganesan A, Maddila C, Murali V et al (2024) Multi-line ai-assisted code authoring. In: *Companion proceedings of the 32nd ACM international conference on the foundations of software engineering*, pp 150–160
- Durrani UK, Akpinar M, Adak MF, Kabakus AT, Ozturk MM, Saleh M (2024) A decade of progress: a systematic literature review on the integration of ai in software engineering phases and activities (2013–2023). *IEEE Access*
- Fagadau ID, Mariani L, Micucci D, Riganelli O (2024) Analyzing prompt influence on automated method generation: an empirical study with copilot. In: *Proceedings of the 32nd IEEE/ACM international conference on program comprehension*, pp 24–34
- Feldman MQ, Anderson CJ (2024) Non-expert programmers in the generative ai future. In: *Proceedings of the 3rd annual meeting of the symposium on human-computer interaction for work*, pp 1–19
- Feng L, Yen R, You Y, Fan M, Zhao J, Lu Z (2024) Coprompt: supporting prompt sharing and referring in collaborative natural language programming. In: *Proceedings of the 2024 CHI conference on human factors in computing systems*, pp 1–21
- Ferdowsi K, Huang R, James MB, Polikarpova N, Lerner S (2024) Validating ai-generated code with live programming. In: *Proceedings of the 2024 CHI conference on human factors in computing systems*, pp 1–8
- Gardella N, Pettit R, Riggs SL (2024) Performance, workload, emotion, and self-efficacy of novice programmers using ai code generation. In: *Proceedings of the 2024 on innovation and technology in computer science education V. 1*. Association for Computing Machinery, pp 290–296

- Gu K, Grunde-McLaughlin M, McNutt A, Heer J, Althoff T (2024) How do data analysts respond to ai assistance? a wizard-of-oz study. In: Proceedings of the 2024 CHI conference on human factors in computing systems, pp 1–22
- Guo J, Mohanty V, Piazzentin Ono JH, Hao H, Gou L, Ren L (2024) Investigating interaction modes and user agency in human-llm collaboration for domain-specific data analysis. In: Extended abstracts of the CHI conference on human factors in computing systems, pp 1–9
- He J, Treude C, Lo D (2025) Llm-based multi-agent systems for software engineering: literature review, vision, and the road ahead. *ACM Trans Softw Eng Methodology* 34(5):1–30
- Hou X, Zhao Y, Liu Y, Yang Z, Wang K, Li L, Luo X, Lo D, Grundy J, Wang H (2024) Large language models for software engineering: a systematic literature review. *ACM Trans Softw Eng Methodology* 33(8):1–79
- Hu X, Xia X, Lo D, Wan Z, Chen Q, Zimmermann T (2022) Practitioners' expectations on automated code comment generation. In: Proceedings of the 44th international conference on software engineering, pp 1693–1705
- Imai S (2022) Is github copilot a substitute for human pair-programming? an empirical study. In: Proceedings of the ACM/IEEE 44th international conference on software engineering: companion proceedings, pp 319–321
- Izadi M, Katzy J, Van Dam T, Otten M, Popescu RM, Van Deursen A (2024) Language models for code completion: a practical evaluation. In: Proceedings of the IEEE/ACM 46th international conference on software engineering, pp 1–13
- Jayagopal D, Lubin J, Chasins SE (2022) Exploring the learnability of program synthesizers by novice programmers. In: Proceedings of the 35th annual ACM symposium on user interface software and technology, pp 1–15
- JetBrains (2024) The state of developer ecosystem 2024: AI insights. <https://www.jetbrains.com/lp/devecosystem-2024/#ai>. Accessed 10 Aug 2025
- Jiang E, Toh E, Molina A, Olson K, Kayacik C, Donsbach A, Cai CJ, Terry M (2022) Discovering the syntax and strategies of natural language programming with generative language models. In: Proceedings of the 2022 CHI conference on human factors in computing systems, pp 1–19
- Jiang S, Coblenz M (2024) An analysis of the costs and benefits of autocomplete in ides. *Proc ACM Softw Eng* 1(FSE):1284–1306
- Kazemitabaar M, Chow J, Ma CKT, Ericson BJ, Weintrop D, Grossman T (2023a) Studying the effect of ai code generators on supporting novice learners in introductory programming. In: Proceedings of the 2023 CHI conference on human factors in computing systems, pp 1–23
- Kazemitabaar M, Hou X, Henley A, Ericson BJ, Weintrop D, Grossman T (2023b) How novices use llm-based code generators to solve cs1 coding tasks in a self-paced learning environment. In: Proceedings of the 23rd Koli calling international conference on computing education research, pp 1–12
- Kazemitabaar M, Williams J, Drosos I, Grossman T, Henley AZ, Negreanu C, Sarkar A (2024a) Improving steering and verification in ai-assisted data analysis with interactive task decomposition. In: Proceedings of the 37th annual ACM symposium on user interface software and technology, pp 1–19
- Kazemitabaar M, Ye R, Wang X, Henley AZ, Denny P, Craig M, Grossman T (2024b) Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In: Proceedings of the 2024 chi conference on human factors in computing systems, pp 1–20
- Kazemitabaar M, Huang O, Suh S, Henley AZ, Grossman T (2025) Exploring the design space of cognitive engagement techniques with ai-generated code for enhanced learning. In: Proceedings of the 30th international conference on intelligent user interfaces, pp 695–714
- Khemka M, Houck B (2024) Toward effective ai support for developers: a survey of desires and concerns. *Commun ACM* 67(11):42–49
- Kitchenham B, Brereton OP, Budgen D, Turner M, Bailey J, Linkman S (2009) Systematic literature reviews in software engineering—a systematic literature review. *Inf Softw Technol* 51(1):7–15
- Kitchenham B, Pretorius R, Budgen D, Brereton OP, Turner M, Niazi M, Linkman S (2010) Systematic literature reviews in software engineering—a tertiary study. *Inf Softw Technol* 52(8):792–805
- Klemmer JH, Horstmann SA, Patnaik N, Ludden C, Burton C Jr, Powers C, Massacci F, Rahman A, Votipka D, Lipford HR et al (2024) Using ai assistants in software development: a qualitative study on security practices and concerns. In: Proceedings of the 2024 on ACM SIGSAC conference on computer and communications security, pp 2726–2740
- Koohestani R, Izadi M (2025) Rethinking ide customization for enhanced hax: a hyperdimensional perspective. In: 2025 IEEE/ACM second IDE workshop (IDE). IEEE, pp 13–15
- Kruse HA, Puhlfürß T, Maalej W (2024) Can developers prompt? a controlled experiment for code documentation generation. In: 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, pp 574–586

- Kuang P, Söderberg E, Höst M (2024) Developers' perspective on today's and tomorrow's programming tool assistance: a survey. In: Companion proceedings of the 8th international conference on the art, science, and engineering of programming, pp 108–116
- Lau S, Guo P (2023) From ""ban it till we understand it"" to ""resistance is futile"": how university programming instructors plan to adapt as more students use ai code generation and explanation tools such as chatgpt and github copilot. In: Proceedings of the 2023 ACM conference on international computing education research-Volume 1, pp 106–121
- Li ZS, Arony NN, Awon AM, Damian D, Xu B (2024) Ai tool use and adoption in software development by individuals and organizations: a grounded theory study. [arXiv:2406.17325](https://arxiv.org/abs/2406.17325)
- Liang JT, Yang C, Myers BA (2024) A large-scale survey on the usability of ai programming assistants: successes and challenges. In: Proceedings of the 46th IEEE/ACM international conference on software engineering, pp 1–13
- Liu C, Zhang X, Zhang H, Wan Z, Huang Z, Yan M (2024) An empirical study of code search in intelligent coding assistant: perceptions, expectations, and directions. In: Companion Proceedings of the 32nd ACM international conference on the foundations of software engineering, pp 283–293
- Manfredi G, Erra U, Gilio G (2023) A mixed reality approach for innovative pair programming education with a conversational ai virtual avatar. In: Proceedings of the 27th international conference on evaluation and assessment in software engineering, pp 450–454
- Mastropaolo A, Pascarella L, Guglielmi E, Ciniselli M, Scalabrino S, Oliveto R, Bavota G (2023) On the robustness of code generation techniques: an empirical study on github copilot. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, pp 2149–2160
- McNutt AM, Wang C, Deline RA, Drucker SM (2023) On the design of ai-powered code assistants for notebooks. In: Proceedings of the 2023 CHI conference on human factors in computing systems, pp 1–16
- Moher D, Liberati A, Tetzlaff J, Altman DG, Group P et al (2010) Preferred reporting items for systematic reviews and meta-analyses: the prisma statement. *Int J Surgery* 8(5):336–341
- de Moor A, van Deursen A, Izadi M (2024) A transformer-based approach for smart invocation of automatic code completion. In: Proceedings of the 1st ACM international conference on AI-powered software, pp 28–37
- Mozannar H, Bansal G, Fournay A, Horvitz E (2024a) Reading between the lines: modeling user behavior and costs in ai-assisted programming. In: Proceedings of the 2024 CHI conference on human factors in computing systems, pp 1–16
- Mozannar H, Bansal G, Fournay A, Horvitz E (2024b) When to show a suggestion? integrating human feedback in ai-assisted programming. In: Proceedings of the AAAI conference on artificial intelligence, vol 38, pp 10137–10144
- Mozannar H, Chen V, Alsobay M, Das S, Zhao S, Wei D, Nagireddy M, Sattigeri P, Talwalkar A, Sontag D (2024c) The realhumaneval: evaluating large language models' abilities to support programmers. [arXiv:2404.02806](https://arxiv.org/abs/2404.02806)
- Nam D, Macvean A, Hellendoorn VJ, Vasilescu B, Myers BA (2023) In-ide generation-based information support with a large language model. *CoRR*
- Nguyen N, Nadi S (2022) An empirical evaluation of github copilot's code suggestions. In: Proceedings of the 19th international conference on mining software repositories, pp 1–5
- OBrien D, Biswas S, Imtiaz SM, Abdalkareem R, Shihab E, Rajan H (2024) Are prompt engineering and todo comments friends or foes? an evaluation on github copilot. In: Proceedings of the IEEE/ACM 46th international conference on software engineering, pp 1–13
- Omidvar Tehrani B, M I, Anubhai A (2024) Evaluating human-ai partnership for llm-based code migration. In: Extended abstracts of the CHI conference on human factors in computing systems, pp 1–8
- Pandey R, Singh P, Wei R, Shankar S (2024) Transforming software development: evaluating the efficiency and challenges of github copilot in real-world projects. [arXiv:2406.17910](https://arxiv.org/abs/2406.17910)
- Pearce H, Ahmad B, Tan B, Dolan-Gavitt B, Karri R (2022) Asleep at the keyboard? assessing the security of github copilot's code contributions. In: 2022 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, pp 754–768
- Peng S, Kalliamvakou E, Cihon P, Demirel M (2023) The impact of ai on developer productivity: evidence from github copilot. [arXiv:2302.06590](https://arxiv.org/abs/2302.06590)
- Penney J, Pimentel JF, Steinmacher I, Gerosa MA (2023) Anticipating user needs: insights from design fiction on conversational agents for computational thinking. In: International workshop on chatbot research and design. Springer, pp 204–219
- Prather J, Reeves BN, Denny P, Becker BA, Leinonen J, Luxton-Reilly A, Powell G, Finnie-Ansley J, Santos EA (2023) "it's weird that it knows what i want": usability and interactions with copilot for novice programmers. *ACM Trans Comput-Human Interact* 31(1):1–31

- Prather J, Reeves BN, Leinonen J, MacNeil S, Randrianasolo AS, Becker BA, Kimmel B, Wright J, Briggs B (2024) The widening gap: the benefits and harms of generative ai for novice programmers. In: Proceedings of the 2024 ACM conference on international computing education research-Volume 1, pp 469–486
- Puryear B, Sprint G (2022) Github copilot in the classroom: learning to code with ai assistance. *J Comput Sci Coll* 38(1):37–47
- Rasnayaka S, Wang G, Shariffdeen R, Iyer GN (2024) An empirical study on usage and perceptions of llms in a software engineering project. In: Proceedings of the 1st international workshop on large language models for code, pp 111–118
- Robe P, Kuttal SK (2022) Designing pairbuddy—a conversational agent for pair programming. *ACM Trans Comput-Human Interact (TOCHI)* 29(4):1–44
- Ross SI, Martinez F, Houde S, Muller M, Weisz JD (2023a) The programmer’s assistant: conversational interaction with a large language model for software development. In: Proceedings of the 28th international conference on intelligent user interfaces, pp 491–514
- Ross SI, Muller M, Martinez F, Houde S, Weisz JD (2023b) A case study in engineering a conversational programming assistant’s persona. [arXiv:2301.10016](https://arxiv.org/abs/2301.10016)
- Ságodi Z, Siket I, Ferenc R (2024) Methodology for code synthesis evaluation of llms presented by a case study of chatgpt and copilot. *Ieee Access* 12:72303–72316
- Sahoo P, Pujar S, Nalawade G, Genhardt R, Mandel L, Buratti L (2024) Ansible lightspeed: a code generation service for it automation. In: Proceedings of the 39th IEEE/ACM international conference on automated software engineering, pp 2148–2158
- Sandoval G, Pearce H, Nys T, Karri R, Garg S, Dolan-Gavitt B (2023) Lost at c: a user study on the security implications of large language model code assistants. In: 32nd USENIX Security Symposium (USENIX Security 23), pp 2205–2222
- Süße T, Kobert M, Grapenthin S, Voigt BF (2023) Ai-powered chatbots and the transformation of work: findings from a case study in software development and software engineering. In: Working conference on virtual enterprises. Springer, pp 689–705
- Senanayake J, Kalutarage H, Petrovski A, Piras L, Al-Kadri MO (2024) Defendroid: real-time android code vulnerability detection via blockchain federated neural network with xai. *J Inf Secur Appl* 82:103741
- Sergeyuk A, Titov S, Izadi M (2024) In-ide human-ai experience in the era of large language models; a literature review. In: Proceedings of the 1st ACM/IEEE workshop on integrated development environments, pp 95–100
- Sergeyuk A, Zakharov I, Koshchenko E, Izadi M (2025) Dataset for the systematic literature review on in-ide human-ai experience. <https://doi.org/10.5281/zenodo.16877797>, version v2
- Shlomov S, Yaeli A, Marreed S, Schwartz S, Eder N, Akrabi O, Zeltyn S (2024) Ida: breaking barriers in no-code ui automation through large language models and human-centric design. [arXiv:2407.15673](https://arxiv.org/abs/2407.15673)
- Spieß C, Gros D, Pai KS, Pradel M, Rabin MRI, Alipour A, Jha S, Devanbu P, Ahmed T (2025) Calibration and correctness of language models for code. In: 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE). IEEE, pp 540–552
- Stack Overflow (2024) 2024 Stack Overflow Developer Survey: AI. <https://survey.stackoverflow.co/2024/ai>. Accessed 10 Aug 2025
- Sun J, Liao QV, Muller M, Agarwal M, Houde S, Talamadupula K, Weisz JD (2022) Investigating explainability of generative ai for code through scenario-based design. In: Proceedings of the 27th international conference on intelligent user interfaces, pp 212–228
- Tan CW, Guo S, Wong MF, Hang CN (2023) Copilot for xcode: exploring ai-assisted programming by prompting cloud-based large language models. [arXiv:2307.14349](https://arxiv.org/abs/2307.14349)
- Tanay BA, Arinze L, Joshi SS, Davis KA, Davis JC (2024) An exploratory study on upper-level computing students’ use of large language models as tools in a semester-long project. [arXiv:2403.18679](https://arxiv.org/abs/2403.18679)
- Tang N, Chen M, Ning Z, Bansal A, Huang Y, McMillan C, Li TJJ (2024) A study on developer behaviors for validating and repairing llm-generated code using eye tracking and ide actions. [arXiv:2405.16081](https://arxiv.org/abs/2405.16081)
- Tian Y, Zhang Z, Ning Z, Li TJJ, Kummerfeld JK, Zhang T (2023) Interactive text-to-sql generation via editable step-by-step explanations. In: EMNLP
- Tian Y, Kummerfeld JK, Li TJJ, Zhang T (2024) Sqlucid: grounding natural language database queries with interactive explanations. In: Proceedings of the 37th annual ACM symposium on user interface software and technology, pp 1–20
- Tong W, Zhang T (2024) Codejudge: evaluating code generation with large language models. In: Proceedings of the 2024 conference on empirical methods in natural language processing, pp 20032–20051
- Vaithilingam P, Zhang T, Glassman EL (2022) Expectation vs. experience: evaluating the usability of code generation tools powered by large language models. In: Chi conference on human factors in computing systems extended abstracts, pp 1–7

- Vaithilingam P, Glassman EL, Groenwegen P, Gulwani S, Henley AZ, Malpani R, Pugh D, Radhakrishna A, Soares G, Wang J et al (2023) Towards more effective ai-assisted programming: a systematic design exploration to improve visual studio intelllicode's user experience. In: 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, pp 185–195
- Vasconcelos H, Bansal G, Fournay A, Liao QV, Wortman Vaughan J (2025) Generation probabilities are not enough: uncertainty highlighting in ai code completions. *ACM Trans Comput-Human Interact* 32(1):1–30
- Vasiliniuc MS, Groza A (2023) Case study: using ai-assisted code generation in mobile teams. In: 2023 IEEE 19th International Conference on Intelligent Computer Communication and Processing (ICCP). IEEE, pp 339–346
- Wang AY, Wang D, Drozdal J, Muller M, Park S, Weisz JD, Liu X, Wu L, Dugan C (2022) Documentation matters: human-centered ai system to assist data science code documentation in computational notebooks. *ACM Trans Comput-Human Interact* 29(2):1–33
- Wang C, Hu J, Gao C, Jin Y, Xie T, Huang H, Lei Z, Deng Y (2023a) Practitioners' expectations on code completion. [arXiv:2301.03846](https://arxiv.org/abs/2301.03846)
- Wang F, Liu X, Liu O, Neshati A, Ma T, Zhu M, Zhao J (2023b) Slide4n: creating presentation slides from computational notebooks with human-ai collaboration. In: Proceedings of the 2023 CHI conference on human factors in computing systems, pp 1–18
- Wang R, Cheng R, Ford D, Zimmermann T (2024) Investigating and designing for trust in ai-powered code generation tools. In: Proceedings of the 2024 ACM conference on fairness, accountability, and transparency, pp 1475–1493
- Wang Z, Zhou Z, Song D, Huang Y, Chen S, Ma L, Zhang T (2025) Towards understanding the characteristics of code generation errors made by large language models. In: 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE). IEEE Computer Society, pp 717–717
- Weber T, Brandmaier M, Schmidt A, Mayer S (2024) Significant productivity gains through programming with large language models. *Proc ACM Human-Comput Interact* 8(EICS):1–29
- Weisz JD, Muller M, Ross SI, Martinez F, Houde S, Agarwal M, Talamadupula K, Richards JT (2022) Better together? an evaluation of ai-supported code translation. In: Proceedings of the 27th international conference on intelligent user interfaces, pp 369–391
- Wermelinger M (2023) Using github copilot to solve simple programming problems. In: Proceedings of the 54th ACM technical symposium on computer science education V. 1, pp 172–178
- Wohlin C (2014) Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th international conference on evaluation and assessment in software engineering, pp 1–10
- Yan L, Hwang A, Wu Z, Head A (2024) Ivie: lightweight anchored explanations of just-generated code. In: Proceedings of the 2024 CHI conference on human factors in computing systems, pp 1–15
- Yen R, Zhu J, Suh S, Xia H, Zhao J (2023) Coladder: supporting programmers with hierarchical code generation in multi-level abstraction. [arXiv:2310.08699](https://arxiv.org/abs/2310.08699)
- Yetistiren B, Ozsoy I, Tuzun E (2022) Assessing the quality of github copilot's code generation. In: Proceedings of the 18th international conference on predictive models and data analytics in software engineering, pp 62–71
- Zhang B, Liang P, Zhou X, Ahmad A, Waseem M (2023) Practices and challenges of using github copilot: an empirical study. [arXiv:2303.08733](https://arxiv.org/abs/2303.08733)
- Zhou H, Li J (2023) A case study on scaffolding exploratory data analysis for ai pair programmers. In: Extended abstracts of the 2023 CHI conference on human factors in computing systems, pp 1–7
- Zhou X, Cao S, Sun X, Lo D (2025a) Large language model for vulnerability detection and repair: literature review and the road ahead. *ACM Trans Softw Eng Methodology* 34(5):1–31
- Zhou X, Liang P, Zhang B, Li Z, Ahmad A, Shahin M, Waseem M (2025b) Exploring the problems, their causes and solutions of ai pair programming: a study on github and stack overflow. *J Syst Softw* 219:112204
- Zhou Y, Zhang H, Huang X, Yang S, Babar MA, Tang H (2015) Quality assessment of systematic reviews in software engineering: a tertiary study. In: Proceedings of the 19th international conference on evaluation and assessment in software engineering, pp 1–14
- Zhu Q, Wang D, Ma S, Wang AY, Chen Z, Khurana U, Ma X (2024) Towards feature engineering with human and ai's knowledge: understanding data science practitioners' perceptions in human&ai-assisted feature engineering design. In: Proceedings of the 2024 ACM designing interactive systems conference, pp 1789–1804
- Ziegler A, Kalliamvakou E, Li XA, Rice A, Rifkin D, Simister S, Sittampalam G, Aftandilian E (2022) Productivity assessment of neural code completion. In: Proceedings of the 6th ACM SIGPLAN international symposium on machine programming, pp 21–29

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Authors and Affiliations

Agnia Sergeyuk¹  · **Ilya Zakharov**¹  · **Ekaterina Koshchenko**²  · **Maliheh Izadi**³ 

✉ Agnia Sergeyuk
agnia.sergeyuk@jetbrains.com

Ilya Zakharov
ilia.zakharov@jetbrains.com

Ekaterina Koshchenko
ekaterina.koshchenko@jetbrains.com

Maliheh Izadi
m.izadi@tudelft.nl

¹ JetBrains Research, Belgrade, Serbia

² JetBrains Research, Amsterdam, The Netherlands

³ Delft University of Technology, Delft, The Netherlands